Predicting Bug fix time: Comparing recurrent neural networks and HMMs

Rakesh Pavan (171IT154)

Department of Information Technology

National Institute of Technology Karnataka, Surathkal

June 2020

Abstract

Large amount of time is spent by software developers in investigating bug reports. It is useful to indicate when a bug report will be closed, since it would help software teams to prioritise their work. Thus, in this work we develop a technique to predict the bug fix time using the temporal activity sequence of the bug given as input. We will make use of the Mozilla Firefox BugZilla database for this purpose. We present an a comparison techniques using RNN, LSTM, GRU and CuDNNGRU and demonstrate the superior performance of using recurrent networks over HMM-based approach used in the existing literature.

Keywords: *Firefox Bugzilla, RNN, LSTM, GRU, CuDNNGRU, HMM*

Contents

1 Introduction

A significant amount of time and manual effort is spent by software developers in investigating bug reports. It is useful to indicate how long it will take for the bug report will be closed, since it would help software teams to prioritise their work. Hence when a bug is found, if it can be figured out that the bug is a severe one, than the respective unit can be assigned a senior developer to fix the bug, other a junior developer can be assigned if the bug not a severe one. Hence it becomes extremely important to find out if the bug is a severe one (i.e will take long time to fix) or a minor one (i.e will take less time to fix).

1.1 Purpose

This project serves the following purposes:

1. Develop a technique to predict the bug fix time using the temporal activity sequence of the bug given as input.
2. Make use of the real-life Mozilla Firefox BugZilla as database for the testing and development.
3. Implement techniques using recurrent networks (namely RNN, LSTM, GRU) and demonstrate their superior performance over existing techniques in literature.

1.2 Intended Audience

The intended audience for this project is large. It could be software developers within an organization, or it could be freelancers. Anyone who works as a programmer or works with significant amount of codes and bugs can make use of this pipeline. However the input to this pipeline is the bug activity sequence, which is more relevant to big organizations with tens of thousands of lines of codes and multiple stages of bug recovery. Hence this pipeline is mainly aimed at big organizations working with big code bases.

1.3 Scope of the project

The scope is limited to finding out whether or not a bug is a severe one (i.e will take long time to fix) or a minor one (can be fixed rather quickly) based on the activity sequence in the early stages of bug reporting and fixing. This early decision can help the organization divide necessary workload into senior or junior developers so as to effectively fix the bug.

2 Requirement Analysis

In this section we analyse various requirements of this project.

2.1 Functional Requirements:

The various functional requirements of this project are:

- Function to scrap raw data from Bugzilla repo.
- Function to analyze statistics from the data.
- Function to process the data to a format usable by ML models.
- Model that can learn from the data and make predictions.
- Functions that measure metrics for the predictions.

2.2 Non-functional Requirements:

The various non-functional requirements of this project are:

- The processing of data should be quick and easy to handle.
- The processed data should be in easy, compact and portable format.
- The model must be able to generalize well to unseen data points.
- The model must take less time to train.
- The model must make quick inferences from given test sample.

3 System Design

In this section we explain and elaborate the designing of our system.

3.1 Design Goals

Following are the goals for designing our system:

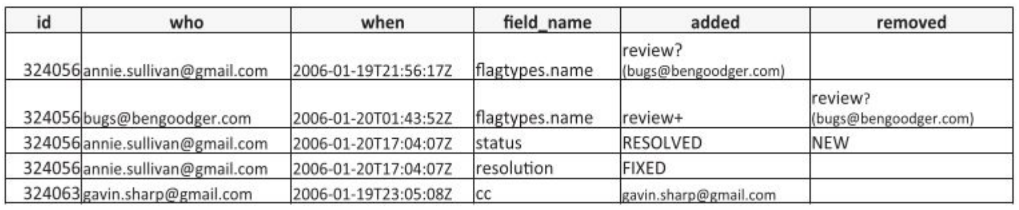
- The design should be easy to read and follow.
- The system should be easy to implement.
- The system must be relatively easy to reproduce and repair in case of bugs.
- The results should be easy to reproduce.
- The design should be compact and must contain all the necessary information only.

3.2 System Architecture

The overall project is divided into 3 phases. The 1st phase is fetching the data and processing the data. This phase also includes analysing the data. The 2nd phase is designing the recurrent neural network model to effectively learn from the previously obtained data. The 3rd and final phase is analysis and testing of results and predictions.

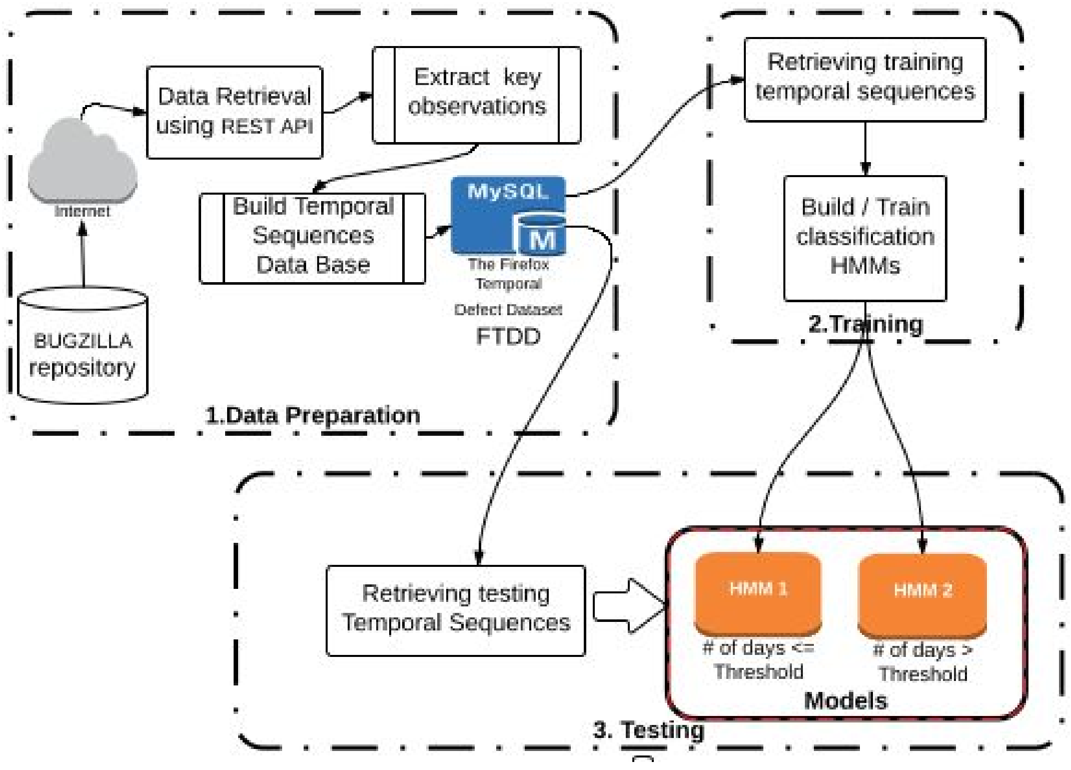
3.2.1 Phase 1:

This phase included collecting, analysis and processing of the data. The collected data is in table format as shown below:



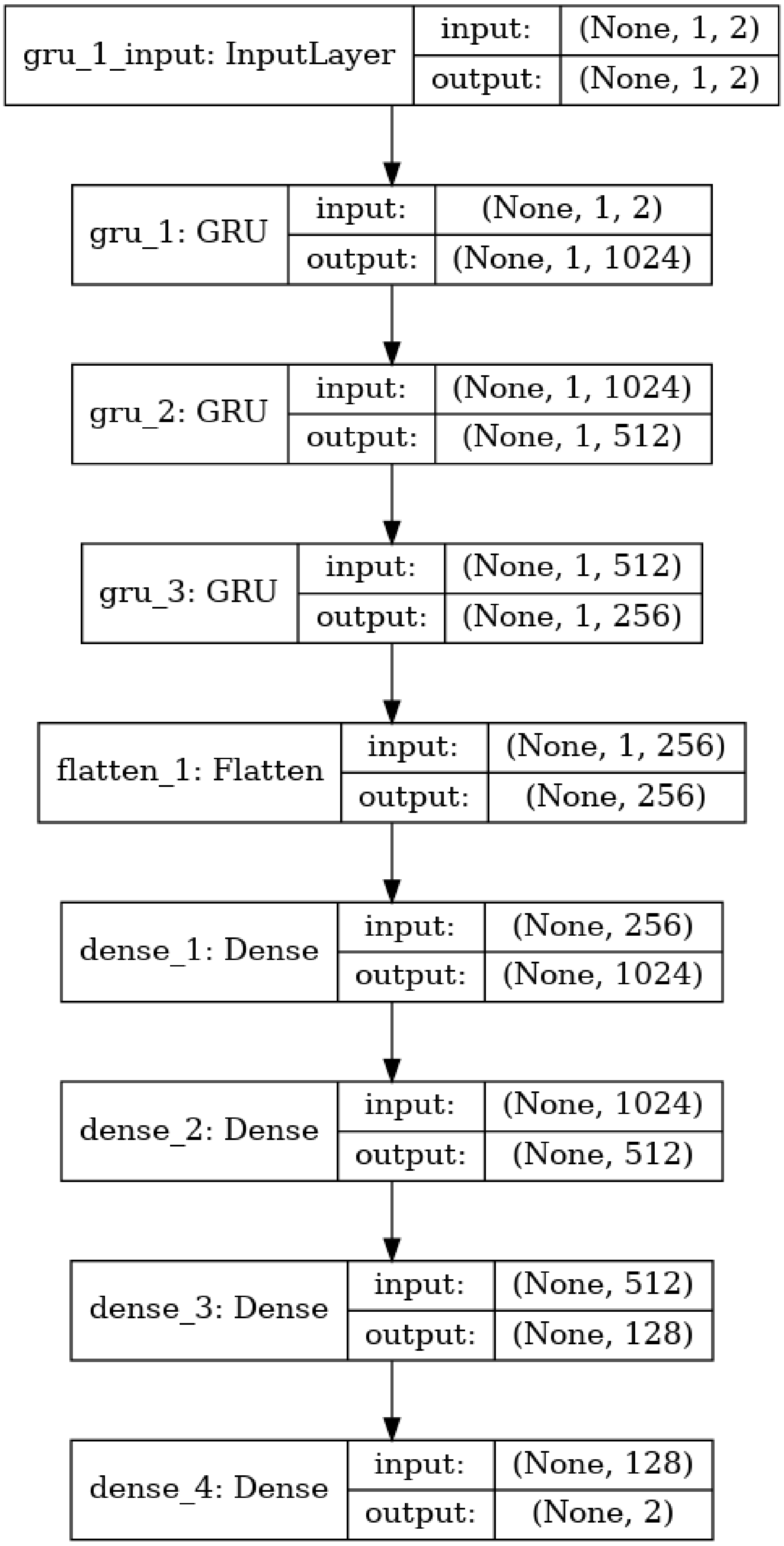
id	who	when	field_name	added	removed
324056	annie.sullivan@gmail.com	2006-01-19T21:56:17Z	flagtypes.name	review? (bugs@bengoodger.com)	
324056	bugs@bengoodger.com	2006-01-20T01:43:52Z	flagtypes.name	review+	review? (bugs@bengoodger.com)
324056	annie.sullivan@gmail.com	2006-01-20T17:04:07Z	status	RESOLVED	NEW
324056	annie.sullivan@gmail.com	2006-01-20T17:04:07Z	resolution	FIXED	
324063	gavin.sharp@gmail.com	2006-01-19T23:05:08Z	cc	gavin.sharp@gmail.com	

The data collection process from the BugZilla repository is summarised in the below figure. For more details refer [1].


Data Retrieval using REST API
Extract key observations
Internet
Build Temporal Sequences Data Base
MySQL
M
The Firefox Temporal Defect Dataset
FTDD
BUGZILLA repository
1.Data Preparation
Retrieving training temporal sequences
Build / Train classification HMMs
2.Training
Retrieving testing Temporal Sequences
HMM 1
# of days <= Threshold
HMM 2
# of days > Threshold
Models
3. Testing

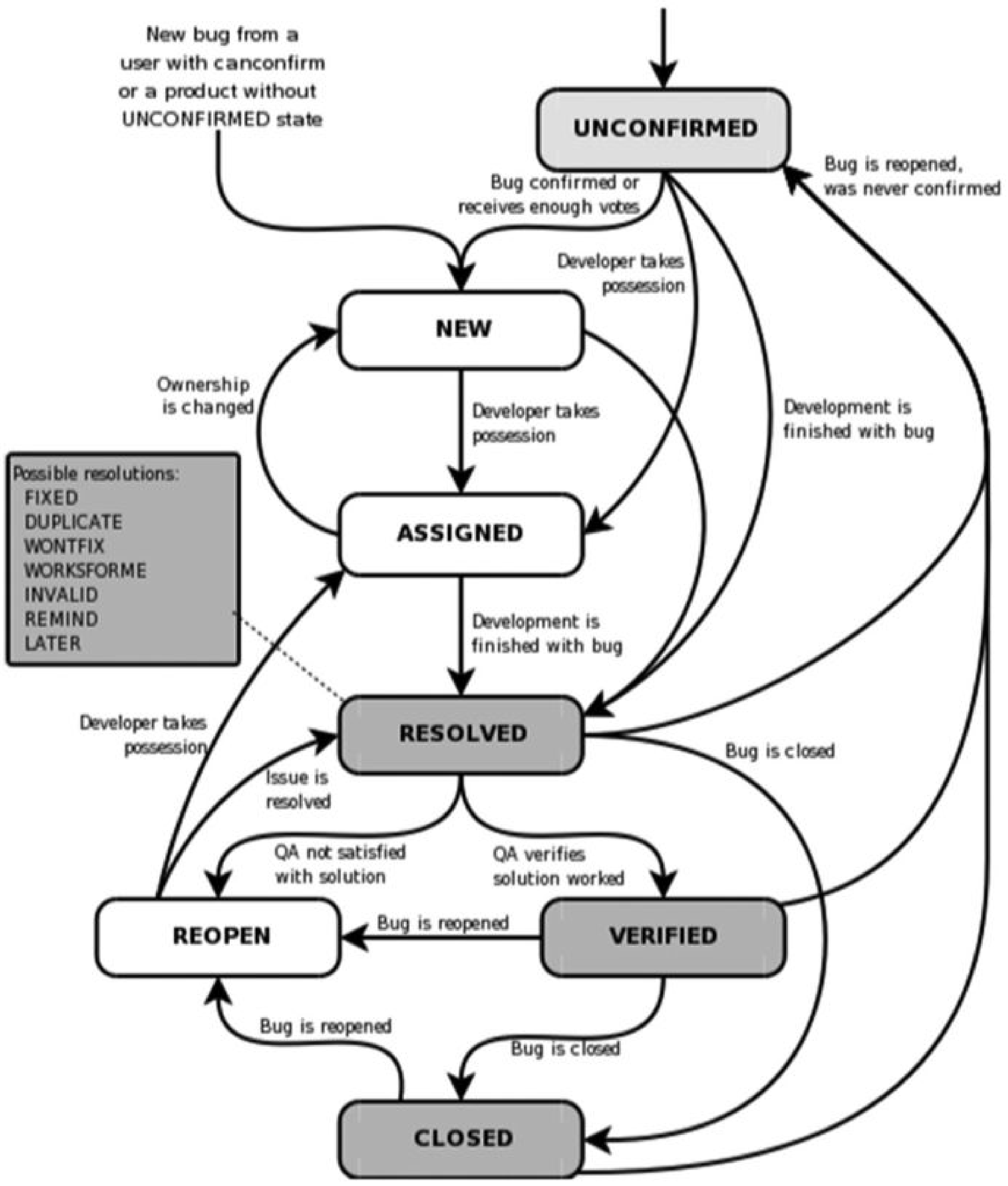
3.2.2 Phase 2:

This phase includes training and building the recurrent model. This was done after repeated testing and analysis with different models and some intuition. The architecture show below is for GRU, however we use the same architecture for other recurrent networks also (i.e LSTM, RNN, CuDNNGRU). The model has 9 layers, out of which 1st one is the Input layer which takes the input data samples. The next 3 layers are GRU cells. Then we use a flatten layer to flatten out all the previous output to a linear vector. Followed by 4 densely connected layers or reducing number of neurons. The final output will be a softmax vector of size 2 which gives the probability distribution of the sample belonging to either the high class or low class, i.e either long time to fix or low time to fix relative to the median value. The architecture finalized for the model is given below:


gru_1_input: InputLayer
input: (None, 1, 2)
output: (None, 1, 2)
gru_1: GRU
input: (None, 1, 2)
output: (None, 1, 1024)
gru_2: GRU
input: (None, 1, 1024)
output: (None, 1, 512)
gru_3: GRU
input: (None, 1, 512)
output: (None, 1, 256)
flatten_1: Flatten
input: (None, 1, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 1024)
dense_2: Dense
input: (None, 1024)
output: (None, 512)
dense_3: Dense
input: (None, 512)
output: (None, 128)
dense_4: Dense
input: (None, 128)
output: (None, 2)

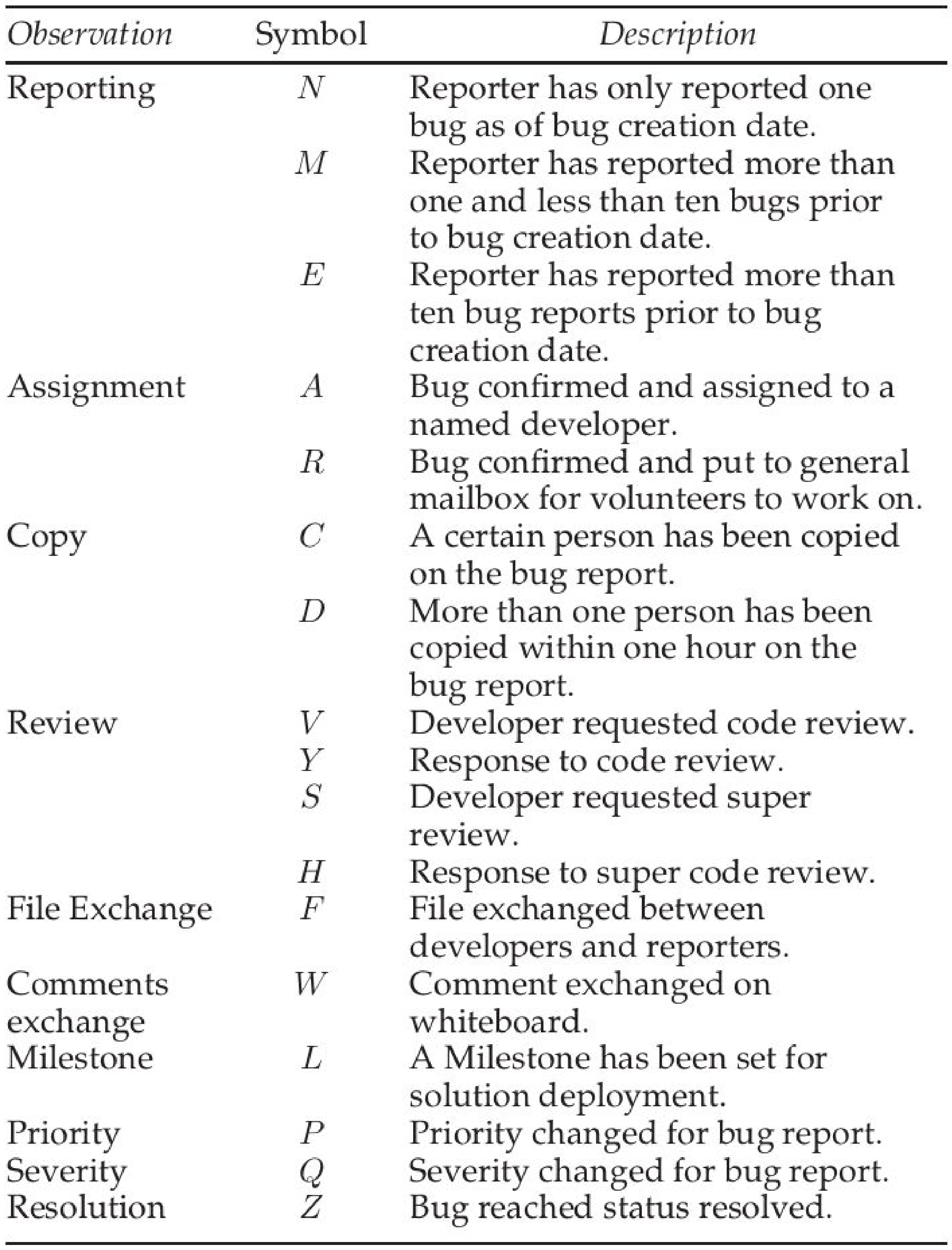
3.3 Detailed Description of Design and Implementation

The bug fix timeline is show in the below figure. It shows how a bug passes through different states in the timeline. This is used to encode a given bug temporal activity sequence into symbolic representation. It covers from the 1st state, i.e when the bug is just discovered, to the final state, i.e when the bug is fixed. A fixed bug can be re-opened also, so there’s an extra re-open state that can lead to a ”new” bug.


New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions: FIXED DUPLICATE WONTFIX WORKSFORME INVALID REMIND LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

3.3.1 Mapping

This section we explain the mapping from each state to a symbol to encode the temporal bug activity in a model-understandable format. The mapping is given below:



Observation	Symbol	*Description*
Reporting	N	Reporter has only reported one bug as of bug creation date.
	M	Reporter has reported more than one and less than ten bugs prior to bug creation date.
	E	Reporter has reported more than ten bug reports prior to bug creation date.
Assignment	A	Bug confirmed and assigned to a named developer.
	R	Bug confirmed and put to general mailbox for volunteers to work on.
Copy	C	A certain person has been copied on the bug report.
	D	More than one person has been copied within one hour on the bug report.
Review	V	Developer requested code review.
	Y	Response to code review.
	S	Developer requested super review.
	H	Response to super code review.
File Exchange	F	File exchanged between developers and reporters.
Comments exchange	W	Comment exchanged on whiteboard.
Milestone	L	A Milestone has been set for solution deployment.
Priority	P	Priority changed for bug report.
Severity	Q	Severity changed for bug report.
Resolution	Z	Bug reached status resolved.

Below is the sample data after encoding in symbolic format:

For bug reports resolved below 60 days		
Sequence of activities	No. of occurrences	Percentage
NCZ	5,679	37.34%
NCCZ	3,131	20.59%
MCZ	1,465	9.63%
NCCCZ	918	6.04%
MCCZ	799	5.25%
ECZ	594	3.91%
NDZ	449	2.95%
ECCZ	380	2.50%
NDCZ	271	1.78%
MCCCZ	241	1.58%
NCCCCZ	233	1.53%
NQCZ	159	1.05%
ECCCZ	137	0.90%
NCDZ	134	0.88%
MDZ	126	0.83%
NCWZ	110	0.72%
NQCCZ	107	0.70%
NCWCZ	98	0.64%
NDCCZ	93	0.61%
NCQCZ	86	0.57%

For bug reports resolved above 60 days		
Sequence of activities	No. of occurrences	Percentage
NWCZ	2,712	15.95%
NCCZ	2,520	14.82%
NCWCZ	2,145	12.62%
NCZ	2,105	12.38%
NCCCZ	1,190	7.00%
NCCWCZ	739	4.35%
MCZ	629	3.70%
MCCZ	585	3.44%
MWCZ	569	3.35%
NCWZ	531	3.12%
NCCCCZ	431	2.54%
NWCCZ	398	2.34%
NWZ	388	2.28%
MCWCZ	365	2.15%
NCCWZ	351	2.06%
ECZ	318	1.87%
MCCCZ	301	1.77%
ECCZ	286	1.68%
NCWCCZ	219	1.29%
NCCCWCZ	217	1.28%

3.3.2 Recurrent Neural Network

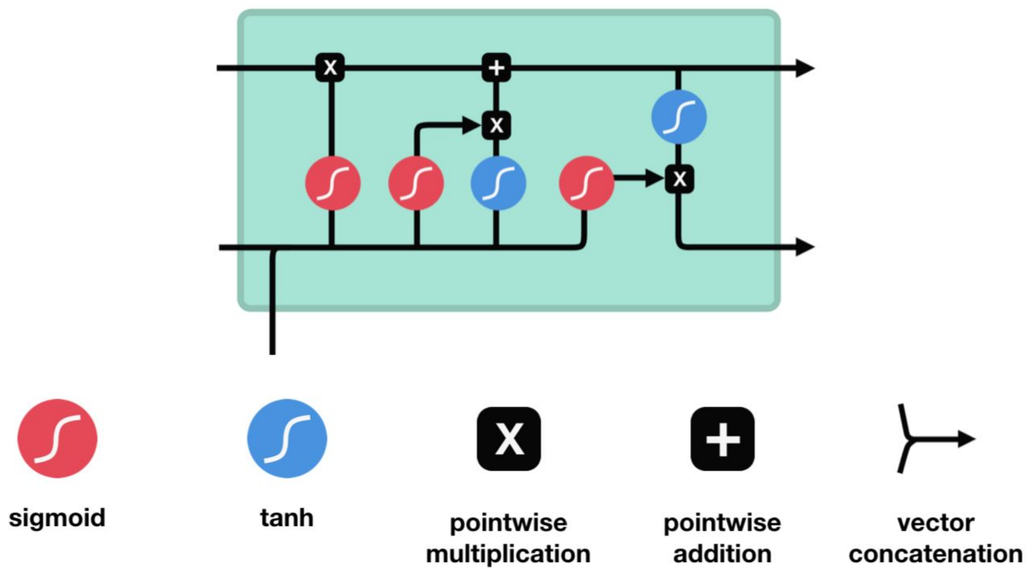
A recurrent neural network has connections between nodes in the form of a directed graph along a temporal sequence. This allows capturing the temporal information from the time series data. Assuming H_t is the RNN state at time t and H_{t-1} is the RNN state at time $t-1$, and the input to the RNN at instant t is X_t , then the RNN can be formulated in the following way:

$$H_t = \sigma_h(W_{hh}H_{t-1} + W_{xh}X_t) \tag{1}$$

Where W_{hh} and W_{xh} are the weights to learn. Learning is done using backpropagation algorithm. Here (and in the following sections) the symbols σ_h respectively is the tanh activation. We train the RNN using our training data and make predictions.

3.3.3 LSTM

LSTM is a modified version of the simple RNN with feedback connections and different gates. The simple RNN has many issues like exploding signal, vanishing gradient problem, etc. and the LSTM tried to solve those issues. LSTM has a forget gate, an input gate, an output gate and a memory cell. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

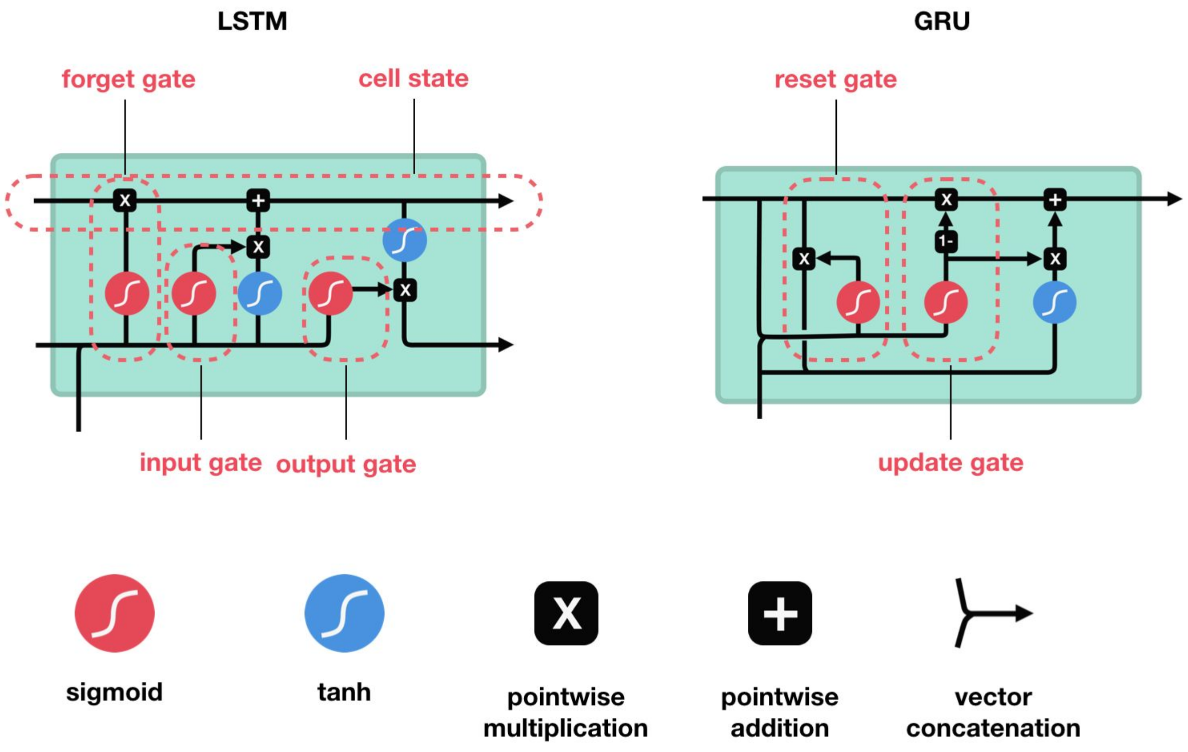


Assuming f_t, i_t, o_t, c_t, h_t represent the forget gate, input gate, output gate, memory cell and the LSTM state at time. Where the terms with W_t, then the LSTM can be formulated as:

$$\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}$$

3.3.4 GRU & CuDNNGRU

The GRU is like a long short-term memory (LSTM) with forget gate but has fewer parameters than LSTM, as it lacks an output gate. The gates in a GRU are called the update Gate and the reset gate. Since the GRU has fewer parameters than an LSTM but has the similar architecture, it can work better incase of smaller datasets. Thus it works better for our dataset consisting of bitcoin prices. A comparison of the LSTM and GRU structure is highlighted in the below figure:



Assuming z_t, r_t, h_t are the update gate, reset gate and current state respectively, then GRU is formulated as (again the weights are learned during training) :

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)
\end{aligned}
$$

The CuDNNGRU is a slightly different and fast GRU implementation backed by CuDNN. It can only be run on GPU, with the TensorFlow backend.

4 External interface requirements

No such external interfaces are required. However the model requires GPU to run, hence it is best to run it on cloud which hosts a GPU server like AWS, GCP, GColab or Kaggle.

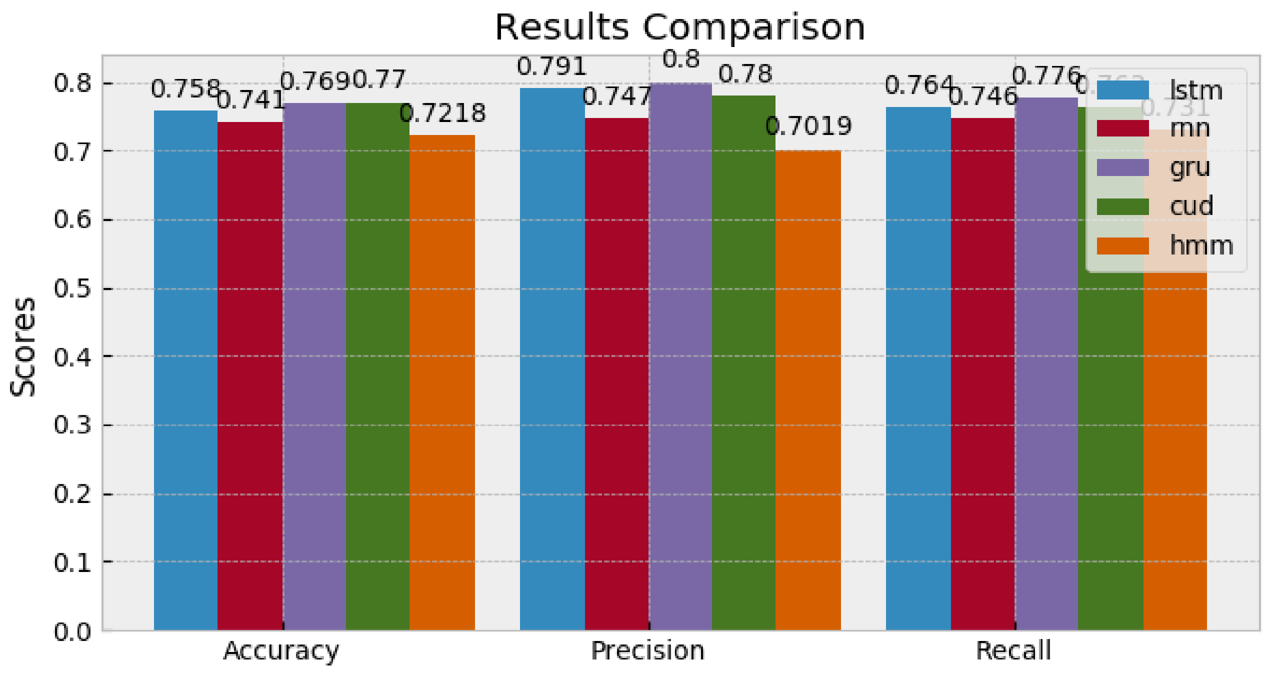
5 System Features

The features are:

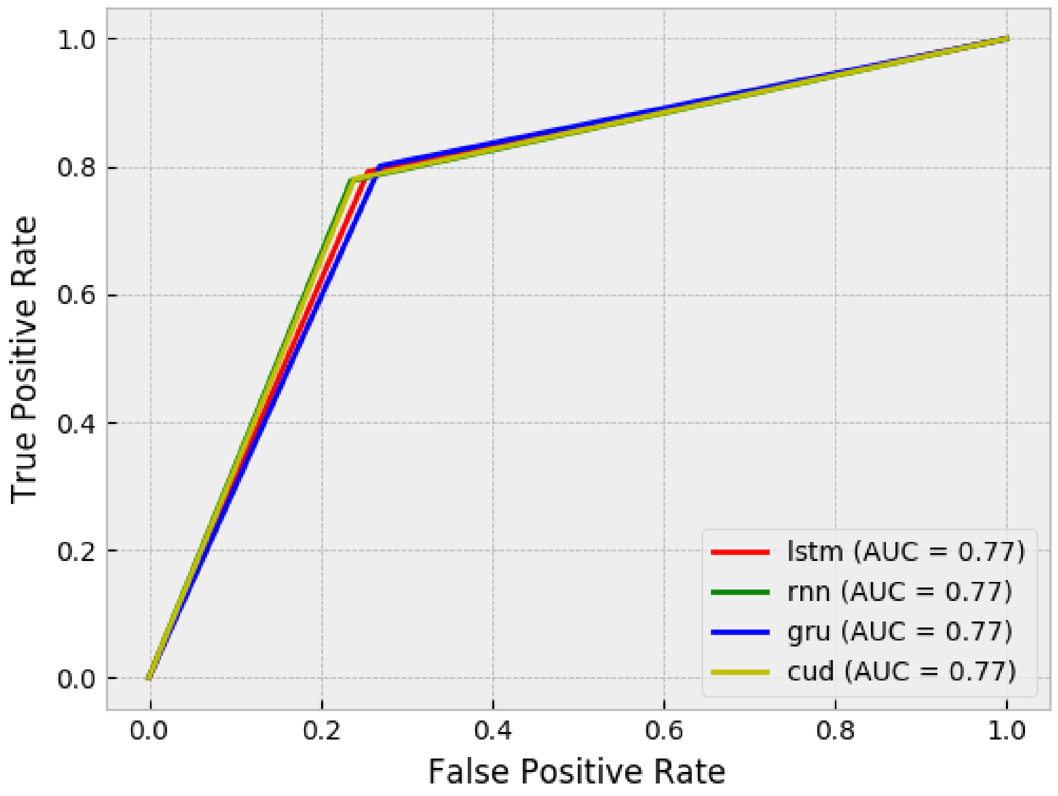
- Scrap raw data from Bugzilla repo.
- Analyze statistics from the data.
- Process the data to a format usable by ML models.
- Model that can learn from the data and make predictions.
- Metrics that measure metrics for the predictions.

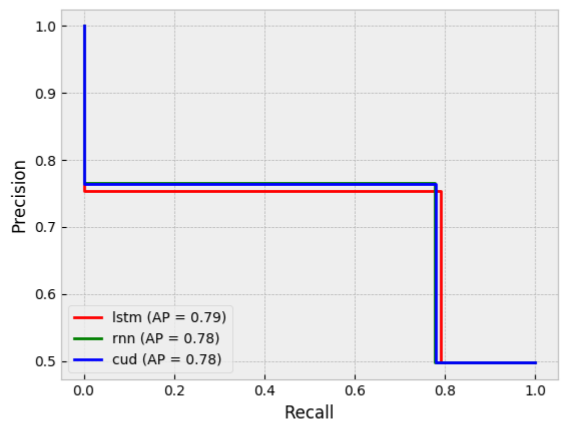
6 Results & Analysis

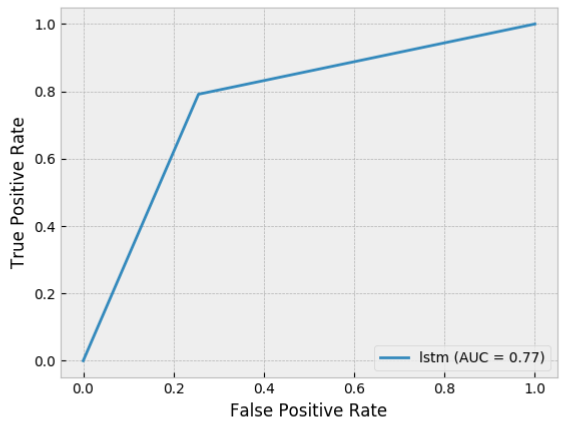
The our method obtains better results are compared to [1]. This is summarised in the below bar graph:

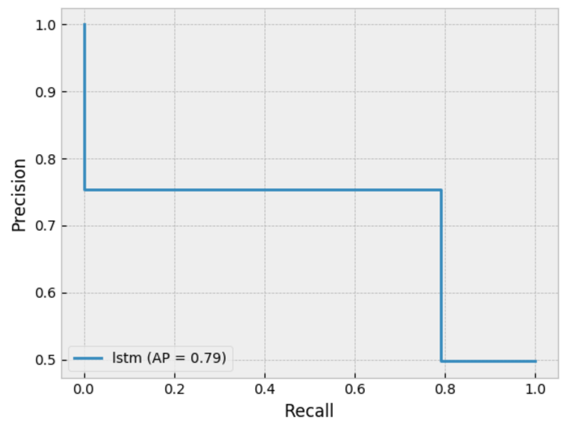


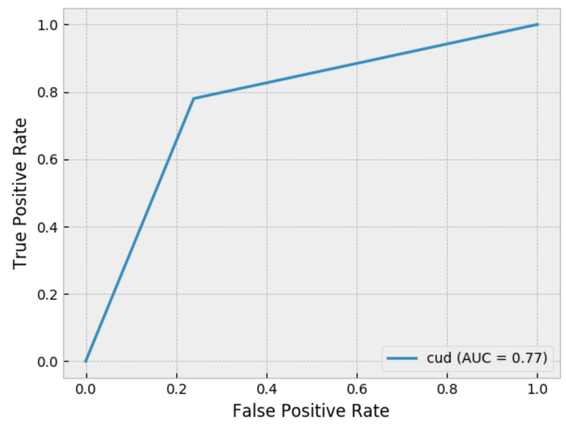
False Positive Rate and True Positive Rate both have values in the range [0, 1]. FPR and TPR both are computed at varying threshold values and a graph is drawn. AUC is the area under the curve of plot FRP vs TPR at different points in [0, 1]. This curve is called the Receiver Operating Characteristic curve.



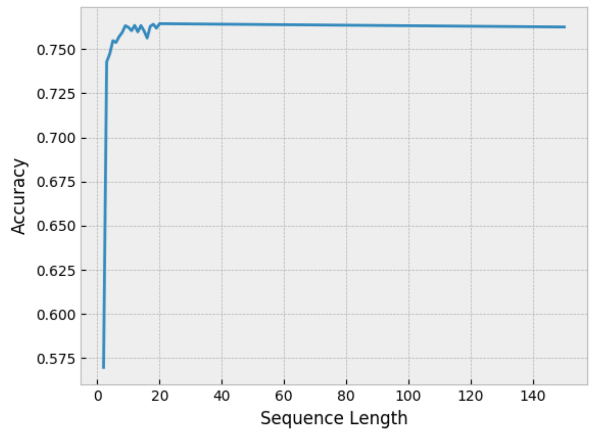


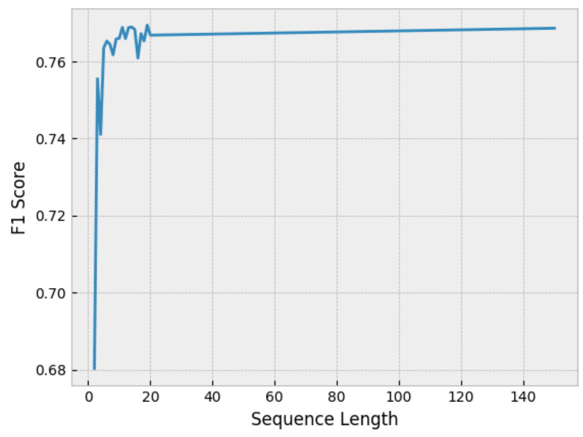




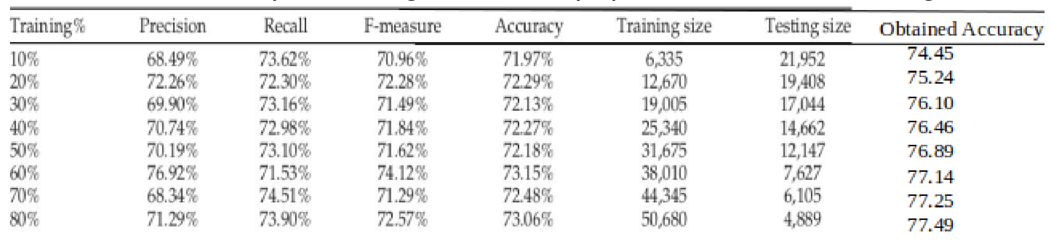


Below is the graph of Accuracy and F1 vs Length of temporal sequence:



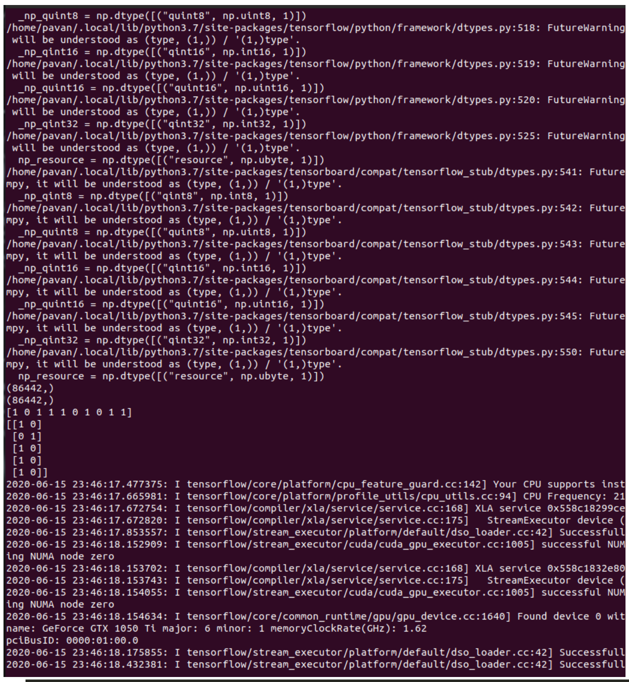


Below is result comparison against base paper for different training %:

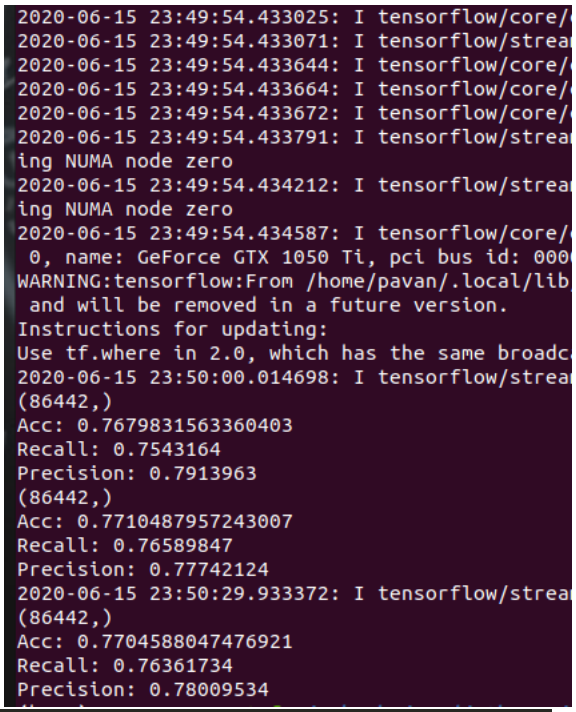


Training%	Precision	Recall	F-measure	Accuracy	Training size	Testing size	Obtained Accuracy
10%	68.49%	73.62%	70.96%	71.97%	6,335	21,952	74.45
20%	72.26%	72.30%	72.28%	72.29%	12,670	19,408	75.24
30%	69.90%	73.16%	71.49%	72.13%	19,005	17,044	76.10
40%	70.74%	72.98%	71.84%	72.27%	25,340	14,662	76.46
50%	70.19%	73.10%	71.62%	72.18%	31,675	12,147	76.89
60%	76.92%	71.53%	74.12%	73.15%	38,010	7,627	77.14
70%	68.34%	74.51%	71.29%	72.48%	44,345	6,105	77.25
80%	71.29%	73.90%	72.57%	73.06%	50,680	4,889	77.49

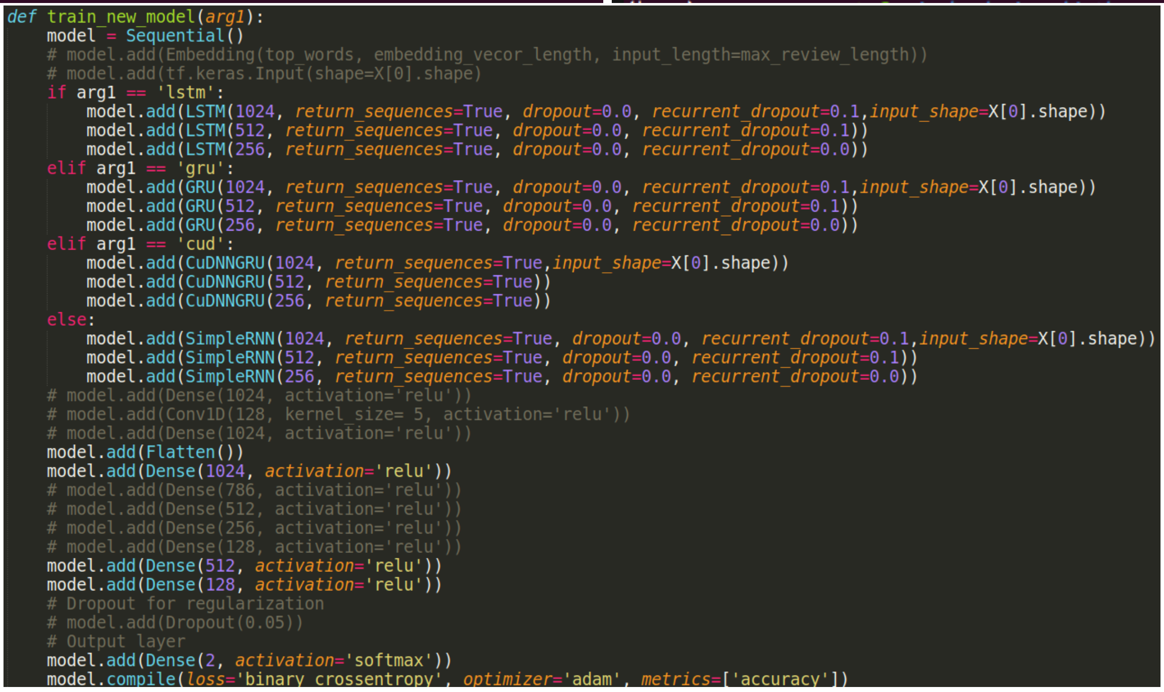
7 Screenshots



```
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:518: FutureWarning
 will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning
 will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:520: FutureWarning
 will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning
 will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: Future
mpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: Future
mpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: Future
mpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: Future
mpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: Future
mpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/pavan/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: Future
mpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
(86442,)
(86442,)
[1 0 1 1 1 0 1 0 1 1]
[[1 0]
 [0 1]
 [1 0]
 [1 0]
 [1 0]]
2020-06-15 23:46:17.477375: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports inst
2020-06-15 23:46:17.665981: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 21
2020-06-15 23:46:17.672754: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x558c18299ce
2020-06-15 23:46:17.672820: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (
2020-06-15 23:46:17.853557: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfull
2020-06-15 23:46:18.152909: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1005] successful NUM
ing NUMA node zero
2020-06-15 23:46:18.153702: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x558c1832e80
2020-06-15 23:46:18.153743: I tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor device (
2020-06-15 23:46:18.154055: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:1005] successful NUM
ing NUMA node zero
2020-06-15 23:46:18.154634: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1640] Found device 0 wit
name: GeForce GTX 1050 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.62
pciBusID: 0000:01:00.0
2020-06-15 23:46:18.175855: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfull
2020-06-15 23:46:18.432381: I tensorflow/stream_executor/platform/default/dso_loader.cc:42] Successfull
```



```
2020-06-15 23:49:54.433025: I tensorflow/core/
2020-06-15 23:49:54.433071: I tensorflow/strea
2020-06-15 23:49:54.433644: I tensorflow/core/
2020-06-15 23:49:54.433664: I tensorflow/core/
2020-06-15 23:49:54.433672: I tensorflow/core/
2020-06-15 23:49:54.433791: I tensorflow/strea
ing NUMA node zero
2020-06-15 23:49:54.434212: I tensorflow/strea
ing NUMA node zero
2020-06-15 23:49:54.434587: I tensorflow/core/
 0, name: GeForce GTX 1050 Ti, pci bus id: 000
WARNING:tensorflow:From /home/pavan/.local/lib
 and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadc
2020-06-15 23:50:00.014698: I tensorflow/strea
(86442,)
Acc: 0.7679831563360403
Recall: 0.7543164
Precision: 0.7913963
(86442,)
Acc: 0.7710487957243007
Recall: 0.76589847
Precision: 0.77742124
2020-06-15 23:50:29.933372: I tensorflow/strea
(86442,)
Acc: 0.7704588047476921
Recall: 0.76361734
Precision: 0.78009534
```



```python
def train_new_model(arg1):
    model = Sequential()
    # model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
    # model.add(tf.keras.Input(shape=X[0].shape)
    if arg1 == 'lstm':
        model.add(LSTM(1024, return_sequences=True, dropout=0.0, recurrent_dropout=0.1,input_shape=X[0].shape))
        model.add(LSTM(512, return_sequences=True, dropout=0.0, recurrent_dropout=0.1))
        model.add(LSTM(256, return_sequences=True, dropout=0.0, recurrent_dropout=0.0))
    elif arg1 == 'gru':
        model.add(GRU(1024, return_sequences=True, dropout=0.0, recurrent_dropout=0.1,input_shape=X[0].shape))
        model.add(GRU(512, return_sequences=True, dropout=0.0, recurrent_dropout=0.1))
        model.add(GRU(256, return_sequences=True, dropout=0.0, recurrent_dropout=0.0))
    elif arg1 == 'cud':
        model.add(CuDNNGRU(1024, return_sequences=True,input_shape=X[0].shape))
        model.add(CuDNNGRU(512, return_sequences=True))
        model.add(CuDNNGRU(256, return_sequences=True))
    else:
        model.add(SimpleRNN(1024, return_sequences=True, dropout=0.0, recurrent_dropout=0.1,input_shape=X[0].shape))
        model.add(SimpleRNN(512, return_sequences=True, dropout=0.0, recurrent_dropout=0.1))
        model.add(SimpleRNN(256, return_sequences=True, dropout=0.0, recurrent_dropout=0.0))
    # model.add(Dense(1024, activation='relu'))
    # model.add(Conv1D(128, kernel_size= 5, activation='relu'))
    # model.add(Dense(1024, activation='relu'))
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    # model.add(Dense(786, activation='relu'))
    # model.add(Dense(512, activation='relu'))
    # model.add(Dense(256, activation='relu'))
    # model.add(Dense(128, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(128, activation='relu'))
    # Dropout for regularization
    # model.add(Dropout(0.05))
    # Output layer
    model.add(Dense(2, activation='softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

8 Conclusion & Future Works

We presented a technique using different kinds of recurrent neural networks for predicting whether a bug takes high recovery time or low recovery time based on the activity sequence in the early stages of bug reporting and fixing. This early decision can help the organization divide necessary workload into senior or junior developers so as to effectively fix the bug. We make use of the Mozilla Firefox BugZilla database for this purpose. We present a comparison techniques using RNN, LSTM, GRU and CuDNNGRU and demonstrate the superior performance of using recurrent networks over HMM-based approach used in the existing literature.

Future works can include:

- Different kinds of preprocessing techniques to extract more effective information from the raw data.
- Using NLP embeddings like Glove or Birt to convert the processed sequence data to vectorized encoding.
- More advanced ML-based pipelines like bi-directional networks along with above encoding.
- More effective strategies to identify bug temporal sequences.
- Converting and treating this task as a regression task (to predict exact number of days) instead of high/low classification task.

9 References

1. M. Habayeb, S. S. Murtaza, A. Miranskyy and A. B. Bener. ”On the Use of Hidden Markov Model to Predict the Time to Fix Bugs”. *in IEEE Transactions on Software Engineering, vol. 44, no. 12, pp. 1224-1244, 1 Dec. 2018*
2. Ardimento, Bilancia, Massimo. “Predicting Bug-Fix Time: Using Standard Versus Topic-Based Text Categorization Techniques”. *Springer International Publishing, Discovery Science 2016*
3. Zhang, Hongyu & Gong, Liang & Versteeg, Steve. (2013). ”Predicting bug-fixing time: An empirical study of commercial software projects”. *Proceedings - International Conference on Software Engineering. 1042-1051/ICSE.2013.6606654*