Project Report Final

Team Members - Javed Habib (12140830), Mohd. Adil (12141080)

Aim of the project

When open-source projects receive contributions and bug reports, they often involve managing a significant number of issues. Labeling issues correctly is essential for efficient issue tracking and resolution. However, it can be a time-consuming task. The aim of this project is to develop a machine learning model that can automatically predict appropriate labels for GitHub issues based on their content.

Progress

About the dataset

It contains 150000 rows with only 2 labels which only classify each row as a bug and not a bug. It contains a body and a title, both of which have mentions and links. Hence appropriate cleaning techniques were used. We used regular expressions and then standard tokenizers to generate embeddings.

```
df.head()
```

	title	body	label	full_text	distilbert-base-nli-mean-tokens_embeddings	albert-base-v1_embeddings	roberta-base-nli-stsb-mean-tokens_embeddings
0	y-zoom piano roll	a y-zoom on the piano roll would be useful.	1	title : y-zoom piano roll body : a y-zoom on t...	[-0.7301139831542969, -0.047545962035655975, 0...	[0.0026986433658748865, 0.12280294299125671, -...	[0.42477285861968994, -0.09550482034683228, -0...
1	buggy behavior in selection	! screenshot from 2016-02-23 21 27 40 https:/...	0	title : buggy behavior in selection body : ! s...	[-0.7117552757263184, 0.05117011442780495, 0.3...	[0.21210505068302155, -0.3419630825519562, -0....	[0.46860456466674805, -0.4723658561706543, 0.3...
2	auto update feature	hi,\r \r great job so far, @saenzramiro ! : \r...	1	title : auto update feature body : hi,\r \r gr...	[-0.545153796672821, 0.023795893415808678, 0.9...	[0.07694720476865768, -0.23390068113803864, -0...	[0.7905717492103577, -0.6955427527427673, 0.66...
3	filter out noisy endpoints in logs	i think we should stop logging requests to:\r ...	1	title : filter out noisy endpoints in logs bod...	[-0.5753872990608215, 0.24625295400619507, 0.5...	[0.14907507598400116, -0.36269834637641907, -0...	[0.34696999192237854, 0.3869624435901642, 0.26...
4	enable pid on / pid off alarm actions for ardu...	expected behavior\r alarm actions pid on and p...	0	title : enable pid on / pid off alarm actions ...	[-0.600375771522522, -0.21507282555103302, 0.5...	[-0.005618194118142128, -0.010725936852395535,...	[0.36912062764167786, 0.15820196270942688, 0.1...

In [9]:

```python
def cleaner(text):
    text = re.sub(r"(@\[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+?", "", text)
    return text

def standardize_accented_chars(text):
    return unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
print(standardize_accented_chars(df['body'].loc[2]))
```

```
hi,\r \r great job so far, @saenzramiro ! : \r \r an auto update feature would be nice to have.\r or alternatively
a menu button to check for the latest version manually.
```

In [11]:

```python
def clean_text(text):
    text = re.sub(r'\r', ' ', text)
    text = re.sub(r'\\r', ' ', text)
    text = re.sub(r'\n', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'@[\w]+', '', text)
    text = re.sub(r'[^A-Za-z0-9\s.,!?]', '', text)
    return text

raw_text = "hi,\\r \\r great job so far, @saenzramiro ! : \\r \\r an auto update feature would be nice to have.
cleaned_text = clean_text(raw_text)
print(cleaned_text)
```

```
hi, great job so far,  !  an auto update feature would be nice to have. or alternatively a menu button to check fo
r the latest version manually.
```

Since the size of large, we had to spend a lot of time in Phase 1 making only the embeddings. We have used 3 models to generate embeddings -

1. distilbert-base-nli-mean-tokens_embeddings (dim - 768)
2. albert-base-v1_embeddings (dim - 768)
3. roberta-base-nli-stsb-mean-tokens_embeddings (dim - 768)

```
model = SentenceTransformer('distilbert-base-nli-mean-tokens')
def generate_embeddings(text):
    embeddings = model.encode(text, convert_to_tensor=False)
    return embeddings.tolist()
# generate_embeddings('I am batman')
df_train['distilbert-base-nli-mean-tokens_embeddings'] = df_train['full_text'].progress_map(generate_embeddings

model = SentenceTransformer('albert-base-v1')
def generate_embeddings(text):
    embeddings = model.encode(text, convert_to_tensor=False)
    return embeddings.tolist()
df_train['albert-base-v1_embeddings'] = df_train['full_text'].progress_map(generate_embeddings)

model = SentenceTransformer('roberta-base-nli-stsb-mean-tokens')
def generate_embeddings(text):
    embeddings = model.encode(text, convert_to_tensor=False)
    return embeddings.tolist()
df_train['roberta-base-nli-stsb-mean-tokens_embeddings'] = df_train['full_text'].progress_map(generate_embeddin


df_train.head()
```

We then tried to cluster these points as it is but the `curse of dimensionality` hit us as it was taking a lot of time and my system kept crashing. Hence we used `PCA` to reduce our vector to 50 dimensions. The dataset size had jumped from `200 MBs` to `8 GBs` after the generation of embeddings etc.
We have tried a lot of clustering methods but only the following gave us good results -

1. DB-SCAN (gave very bad results)
2. Agglomerative Hierarchical Clustering
3. Spectral Clustering
4. K means

Then I manually the clusters by sampling and identified the following potential new classes -

1. Features
2. Bugs
3. Auth
4. Maintenance
5. Security
6. Testing

Experiments

We tried to lable our data through various LLMs by ujsing prompt engineering and `langchain` but it was giving very poor results and took a lot of time to process.

Labelling using prompts

In [40]:
```
from langchain.llms import LlamaCpp
from langchain.document_loaders import PyPDFLoader
from langchain.embeddings import LlamaCppEmbeddings
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.document_loaders import TextLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.text_splitter import RecursiveCharacterTextSplitter
import os
from langchain.document_loaders import PyPDFLoader
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain.llms import LlamaCpp
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain.chains import LLMChain
from langchain.llms import GPT4All
from langchain.prompts import PromptTemplate
import warnings
warnings.filterwarnings('ignore')
```

In [72]:
```
df['text_full'] = df['cleaned_title'] + ' ' + df['cleaned_body']
```

In [75]:
```
template = """
Issue text : {text}
Question: {question}

Answer: Label : """

prompt = PromptTemplate(template=template, input_variables=['doctext',"question"])
callback_manager = CallbackManager([StreamingStdOutCallbackHandler()])
# Make sure the model path is correct for your system!
llm = LlamaCpp(
    model_path="../LRNLP/llama-2-7b-chat.Q4_K_M.gguf",
    temperature=0.75,
    max_tokens=200000000,
    top_p=1,
    callback_manager=callback_manager,
    verbose=True,  # Verbose is required to pass to the callback manager
)
```

In [76]:
```
text = """
title : tighten up cli parsing body : i have a multilanguage project, so i want to organize my tests inside a ma
"""
question = """Please assign only one label from teh following labels
1. Features
2. Bugs
3. Auth
4. Maintenance
5. Security
6. Testing
make sure not to differ from these classes. The output should only be one word"""
prompt = f"""
Issue text : {text}
Question: {question}

Answer: Label :
"""

llm(prompt)
```

```
Testing
```

```
llama_print_timings:        load time =     204.29 ms
llama_print_timings:      sample time =       0.51 ms /     3 runs   (    0.17 ms per token,  5917.16 tokens per s
econd)
llama_print_timings: prompt eval time =    3645.50 ms /   165 tokens (   22.09 ms per token,    45.26 tokens per s
econd)
llama_print_timings:        eval time =     112.87 ms /     2 runs   (   56.43 ms per token,    17.72 tokens per s
econd)
llama_print_timings:       total time =    3785.41 ms
```

Out[76]: 'Testing'

It is giving good labels...so now we can give it some data annotation task and proceed with semi supervised learning.

In [84]:
```
subset = df.sample(100, random_state=69)
subset.head()
```

Initially it was doing well but as we went on and on, the performance degreaded and hence we got the following results.

```
In [87]:  1 llm_annotations
```

```
 'Bugs',
 'Security\n\nExplanation: The given text mentions the idea of extracting data generation code into an independe
nt library, which is a common practice in software development to reuse and maintain code. The issue being discu
ssed is labeled as "Security", indicating that it pertains to issues related to security in the context of softw
are development.',
 'The GitHub issue has the label "Features".',
 'Bugs',
 'The label of the given GitHub issue is "Bugs".',
 'The given GitHub issue has the label "Bugs".',
 '"Bugs"',
 'Security',
 '035 localid is the value of tag 035 and code a when code 9 value is inspire example marcdatafield tag 035 ind1
ind2 marcsubfield code 9 inspiremarcsubfield m',
 'Bugs',
 'Security\n    \nExplanation: The given GitHub issue mentions that the job is running on container-based infras
tructure, which does not allow use of sudo, setuid, and setguid executables. This indicates that the issue is re
lated to security, hence the label is "Security".',
 'Bugs\n\nExplanation: The given text mentions "add measurement form ability to add measurement for each type of
measurement" which is a feature request. Therefore, the label for this GitHub issue would be "Bugs".',
```

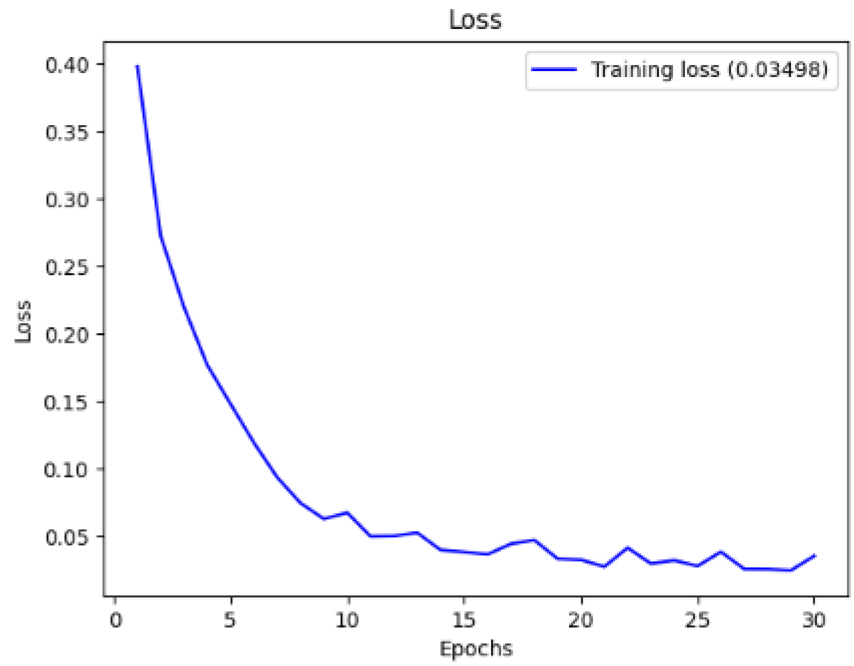
As we can see, there is a class bias as it is labelling everything as Bug or a security threat.
This is mainly because some of these very few of these issues are in spanish, some are incomplete and have mentions. More importantly, the bias was where the users were mentioning specific parts of the code which had no mneomic attribute.

Finally we had to go with `manual annotation of 150 texts` and we got to teh following labels for the cluster results of Agglomerative Clustering.

Cluster ID	Name
0	Testing
1	Bug
2	Security
3	Auth
4	Feature
5	Maintainence

We then learned a model using these labels obtained from clustering and took trained a model.

```
accuracy: 0.8713
Epoch 12/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0498 - accuracy: 0.9820 - val_loss: 0.6029 - val_
accuracy: 0.8728
Epoch 13/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0522 - accuracy: 0.9815 - val_loss: 0.6457 - val_
accuracy: 0.8698
Epoch 14/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0395 - accuracy: 0.9858 - val_loss: 0.5885 - val_
accuracy: 0.8763
Epoch 15/30
750/750 [==============================] - 5s 7ms/step - loss: 0.0379 - accuracy: 0.9865 - val_loss: 0.6870 - val_
accuracy: 0.8732
Epoch 16/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0362 - accuracy: 0.9876 - val_loss: 0.7413 - val_
accuracy: 0.8733
Epoch 17/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0440 - accuracy: 0.9846 - val_loss: 0.7848 - val_
accuracy: 0.8718
Epoch 18/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0468 - accuracy: 0.9835 - val_loss: 0.7439 - val_
accuracy: 0.8792
Epoch 19/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0329 - accuracy: 0.9892 - val_loss: 0.7923 - val_
accuracy: 0.8687
Epoch 20/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0321 - accuracy: 0.9887 - val_loss: 0.7822 - val_
accuracy: 0.8700
Epoch 21/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0271 - accuracy: 0.9907 - val_loss: 0.8814 - val_
accuracy: 0.8672
Epoch 22/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0410 - accuracy: 0.9860 - val_loss: 0.8969 - val_
accuracy: 0.8708
Epoch 23/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0293 - accuracy: 0.9906 - val_loss: 1.0043 - val_
accuracy: 0.8635
Epoch 24/30
750/750 [==============================] - 5s 7ms/step - loss: 0.0317 - accuracy: 0.9894 - val_loss: 0.9242 - val_
accuracy: 0.8707
Epoch 25/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0275 - accuracy: 0.9906 - val_loss: 1.0028 - val_
accuracy: 0.8673
Epoch 26/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0380 - accuracy: 0.9877 - val_loss: 0.9444 - val_
accuracy: 0.8698
Epoch 27/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0253 - accuracy: 0.9924 - val_loss: 1.0067 - val_
accuracy: 0.8653
Epoch 28/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0252 - accuracy: 0.9920 - val_loss: 0.9689 - val_
accuracy: 0.8697
Epoch 29/30
750/750 [==============================] - 3s 5ms/step - loss: 0.0243 - accuracy: 0.9914 - val_loss: 1.0763 - val_
accuracy: 0.8668
Epoch 30/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0350 - accuracy: 0.9893 - val_loss: 1.1159 - val_
accuracy: 0.8633
```



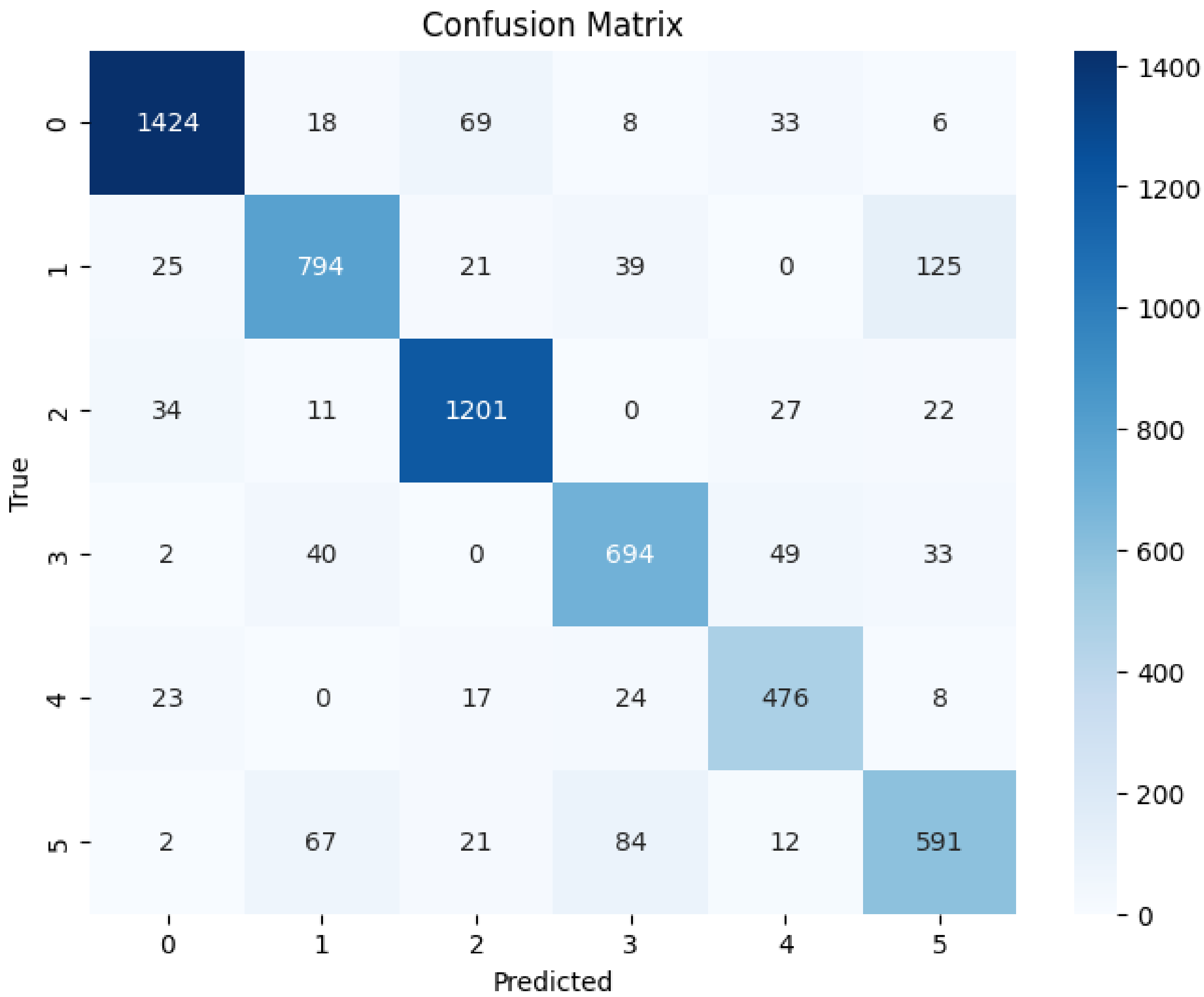

```python
from sklearn.metrics import classification_report
y_pred = model.predict(X_test)
y_pred_labels = tf.argmax(y_pred, axis=1).numpy()
print(classification_report(y_test, y_pred_labels))
```

```
188/188 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.94      0.91      0.93      1558
           1       0.85      0.79      0.82      1004
           2       0.90      0.93      0.92      1295
           3       0.82      0.85      0.83       818
           4       0.80      0.87      0.83       548
           5       0.75      0.76      0.76       777

    accuracy                           0.86      6000
   macro avg       0.84      0.85      0.85      6000
weighted avg       0.86      0.86      0.86      6000
```

```
188/188 [==============================] - 0s 2ms/step
```



Conclusion

We learnt the following things from our project.

1. The concept of cluster based semi-supervised learning.
2. hands on use of PCA and tsne to make cluster. (Did a lot of experimentation there)
3. Various types of clustering techniques and how they work.
4. Made a benchmarking code but it did not work well as sillouhete score is not valid for non-overlapping clusters
5. The way people point out issues on github is very ambiguous and so Encoders liek BERT, Alberta and Roberta are unable to capture that.
6. Implemented Langchain to annotation but LLMs hallucinate a lot are unable to perform well.
7. Manually labelled the clusters and got the model accuracy of `86 percent` (approx.)
   Please find attached my code and other files.