Chatbot for Intent Recognition

Report by

Rehapriadarsini Manikandasamy

The George Washington University

Author Note

This report was prepared for DATS 6312, taught by Professor Amir Jafari

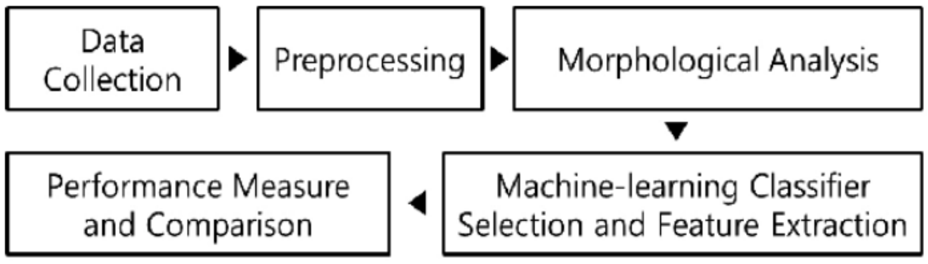
TABLE OF CONTENTS

INTRODUCTION

Intent recognition is sometimes called as intent classification is the task of taking a written or spoken input, and classifying it based on what the user wants to achieve. Intent recognition forms an essential component of chatbots and finds use in sales conversions, customer support, and many other areas. Intent recognition is a form of natural language processing (NLP), a subfield of artificial intelligence. NLP is concerned with computers processing and analyzing natural language, i.e., any language that has developed naturally, rather than artificially, such as with computer coding languages. Intent recognition works through the process of providing examples of text alongside their intents to a machine learning (ML) model. This is known as using training data to train a model.

Chatbots use natural language processing (NLP) to understand the user's intent which means recognizing user's aim in starting any conversation. Intent recognition is a critical feature in chatbot architecture that determines if a chatbot will succeed at fulfilling the user's needs in sales, marketing, or customer service. The quantity of the chatbot's training data is key to maintaining a good conversation with the users. However, the data quality determines the bot's ability to detect the right intent and generate the correct response. Natural language processing (NLP) allows the chatbot to understand the user's message, and machine learning classification algorithms to classify this message based on the training data, and deliver the correct response. The chatbots' intent detection component helps identify what general task or goal the user is trying to accomplish to handle the conversation with different strategies. Once the goal is known, the bot must manage a dialogue to achieve that goal, ensuring that follow-up questions are handled correctly. The intent detector uses a text classifier to classify the input sentence into one of several classes.

One such major application of intent recognition is Google's Dialog Flow which is been used as Malaysian Airlines, Dominos etc. in order to understand their customers and improve their sales.


Data Collection
Preprocessing
Morphological Analysis
Machine-learning Classifier Selection and Feature Extraction
Performance Measure and Comparison

DESCRIPTION OF DATASET

The motivation for the Intent Recognition dataset has been drawn from the Natural Language Understanding Benchmark incorporating Few-Shot-Detection. Few-Shot-Intent-Detection is a repository designed for few-shot intent detection with/without Out-of-Scope (OOS) intents. It includes popular challenging intent detection datasets and baselines. Here is the basic understanding of OOD-OOS and ID-OOS intents.

OOD-OOS: i.e., out-of-domain OOS. General out-of-scope queries which are not supported by the dialog systems, also called out-of-domain OOS. For instance, requesting an online NBA/TV show service in a banking system.

ID-OOS: i.e., in-domain OOS. Out-of-scope queries which are more related to the in-scope intents, which makes the intent detection task more challenging. For instance, requesting a banking service that is not supported by the banking system.

The scope of our dataset has been inherited from 'CLINC150' intent dataset and since our purpose was to built a chatbot for intent recognition we have developed a custom-built dataset.

Features of the dataset are described as:

Tags: Intents

Patterns: User Input

Responses: Bot Responses

The above-described features are viewed as:

{"intents":

[{"tag": "greeting",

"patterns": ["Hi", "How are you", "Is anyone there?", "Hello", "Good day"],

"responses": ["Hello, thanks for visiting", "Good to see you again", "Hi there, how can I help?"],

]}}

DESCRIPTION OF WORK

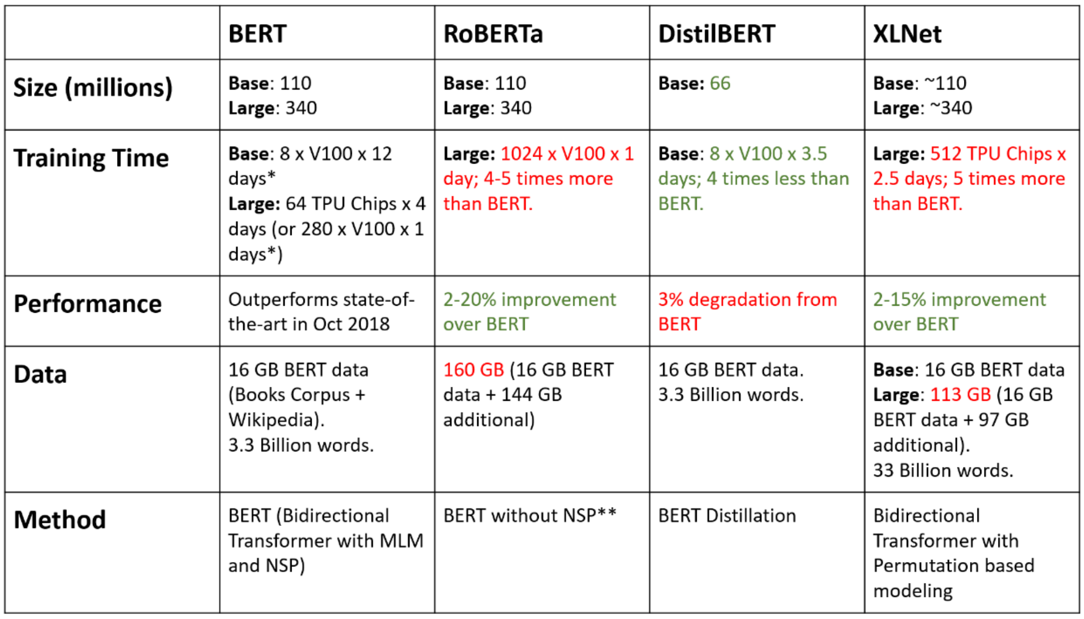
My contribution to this project is to develop Distil BERT model. And GUI for chatbot was developed.

Information on Algorithm development:

DISTILBERT model of the project was inspired based on the following background study.

Distil BERT:

As Transfer Learning from large-scale pre-trained models becomes more prevalent in Natural Language Processing (NLP), operating these large models in on-the-edge and/or under constrained computational training or inference budgets remains challenging. Hence a method has been proposed to pre-train a smaller general-purpose language representation model, called Distil BERT, which can then be fine-tuned with good performances on a wide range of tasks like its larger counterparts. While most prior work investigated the use of distillation for building task-specific models, we leverage knowledge distillation during the pre-training phase and show that it is possible to reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster. To leverage the inductive biases learned by larger models during pre-training, we introduce a triple loss combining language modeling, distillation and cosine-distance losses. Distill BERT is smaller, faster and lighter model is cheaper to pre-train and demonstrates its capabilities for on-device computations in a proof-of-concept experiment and a comparative on-device study.



	BERT	**RoBERTa**	**DistilBERT**	**XLNet**
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base:** 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.	**Large:** 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

EXPERIMENTAL SETUP

AWS_GCP_SETUP:

AWS instance setup:

1. Log in to AWS account and launch console.
2. Launch virtual machine
3. Select AMI if already exists
4. GPU – g3.4xlarge
5. Generate a key pair and download the key in .pem version
6. Launch the instance

GCP instance setup:

7. Launch Google cloud console
8. Launch compute engine
9. Create a project and click create intents
10. Select the following options in the instance creation tab:
11. Zone – US central-a-zone
12. Series- N1
13. Machine type – n1-standard-8(30GB)
14. GPU – nvidia-teslaT4/P4
15. Boot disk – ubuntu, version – 20.04, size – 300
16. Add the public key generated using Puttygen software

For windows,

17. Create a SSH session in mobaxterm
18. Host: External IP from any one of the instances
19. Name: ubuntu
20. Add the private key downloaded from the instances
21. Click ok.
22. Create a folder in the cloud

PyCharm integration:

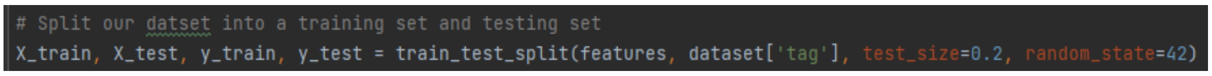
23. Create a project
24. Tools -> Deployment -> Configuration -> Add SFTP
25. SSH configuration: Host- IP address from instances, name: ubuntu, authorize using the private key pair.
26. Test the connection
27. In mappings, map the remote local path and the cloud folder's path
28. Deployment -> Configuration -> Automatic upload
29. Setup the Interpreter
30. Python interpreter -> SSH interpreter -> Existing server configuration -> Choose the cloud -> click ok.

DATASET

Since the dataset is custom built for our use, we don’t have test data we have implemented .json file and converted into the csv file. The actual size of the train data is 868 rows and 3 attributes we are using the same data and mark it as test data to test the performance of the model. Hence before feeding the data to the models.

For DISTIL BERT: we are using the entire train dataset and then by using train -test-split dividing the data into train and test with the test size of 0.2.

```
##Loading the json file:
with open('/home/ubuntu/TERM_PROJECT/intents.json') as json_file:
    data = json.load(json_file)
    print(data)
data = data['intents']


#Reading the json file and converting it into the dataframe:
dataset = pd.DataFrame(columns=['tag', 'patterns', 'responses'])
for i in data:
    tag = i['tag']
    for t, r in zip(i['patterns'], i['responses']):
        row = {'tag': tag, 'patterns': t, 'responses':r}
        dataset = dataset.append(row, ignore_index=True)
print(dataset['tag'].head(250))
```



```
# Split our datset into a training set and testing set
X_train, X_test, y_train, y_test = train_test_split(features, dataset['tag'], test_size=0.2, random_state=42)
```

MODEL IMPLEMENTATION:

Distil BERT

For Distil BERT implementation we are making use of distilled-bert-uncased model

```
#Defining the pretrained distil bert model
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel,
ppb.DistilBertTokenizer, 'distilbert-base-uncased')
# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

Once the data has been tokenized by using the pattern column of the data, we are using attention mask and building a tensor out of the padded input and sending it to the distil bert model

```
tokenized = dataset['patterns'].apply((lambda x: tokenizer.encode(x,
add_special_tokens=True)))
max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
np.array(padded).shape
attention_mask = np.where(padded != 0, 1, 0)
attention_mask.shape
```

```
#create an input tensor out of the padded token matrix, and send that to 
DistilBERT
input_ids = torch.tensor(np.array(padded))
with torch.no_grad():
  # last_hidden_states holds the outputs of DistilBERT.
  # It is a tuple with the shape (number of examples, max number of tokens 
in the sequence, number of hidden units in the DistilBERT model).
    last_hidden_states = model(input_ids)
 # Slice the output for the first position for all the sequences, take all 
hidden unit outputs
features = last_hidden_states[0][:,0,:].numpy()
```

We have used RandomizedSearchCV where the estimator is Logistic Regression to find the best params and scores for our model since we are using Distil BERT to embed the text and modelling is performed by Logistic Regression

```
# search for the best value of the C parameter, which determines 
regularization strength.
parameters = {'C': np.linspace(0.0001, 100, 20)}

lr_finder = RandomizedSearchCV(estimator = LogisticRegression() , scoring = 
'accuracy',
                               param_distributions = parameters,
                               cv = KFold(n_splits=5, shuffle=True, 
random_state = seed),
                               verbose=50, random_state=seed, n_jobs = -1)
lr_finder.fit(X_train, y_train)

print('best parameters: ', lr_finder.best_params_)
print('best scores: ', lr_finder.best_score_)
```

The models performance is measured by using the classification report from sklearn.metric package.

```
lr_cv = lr_finder.best_estimator_.fit(X_train, y_train)
y_lr_pred = lr_cv.predict(X_test)
print(metrics.classification_report(y_test, y_lr_pred))
```

CHATBOT:

The packages required for the chatbot implementation are tkinter and python. Steps to set up tkinter on AWS are as follows:

1.First we need to start the AWS instance and connect to pycharm

2.install XQuartz terminal,

3.Connect to the terminal using the followiing command - ssh -X -i file.pem ubuntu@publickey

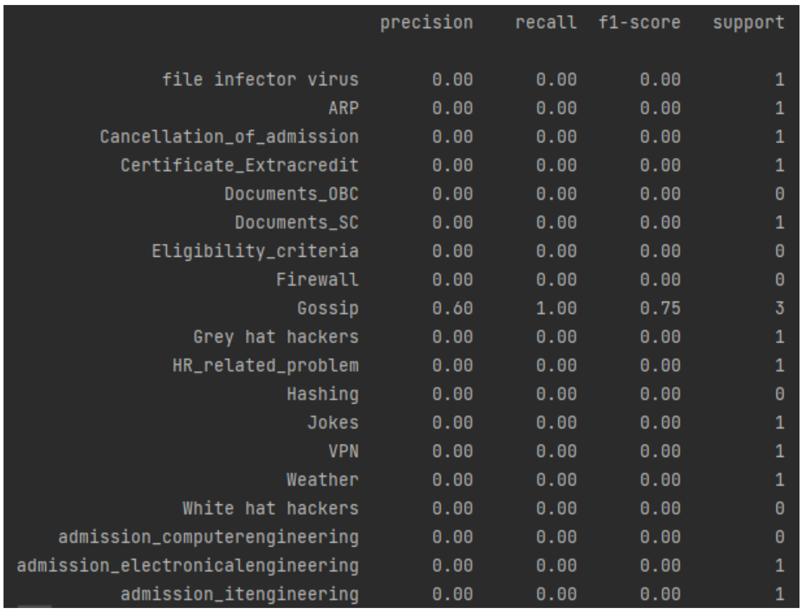
4.run the main.py first then run the chat_bot.py file

The layout of the UI is divided into different functions we have created a class named Chat Application we firstly have init function which is use to initialize the Chat Application and run function shows the window up and running , Then after we have created a function setup_main_window which forms the major functionalities of the chat window like head label is used to give a label to the head of the chat window, tiny label is used on the user end to display the message for eg " hello , how are you ?" tiny divider is used to have a break between the chat window and header. Scrollbar is used to scroll through the messages in the chat application, message entry box is where the user inputs the message and send button.

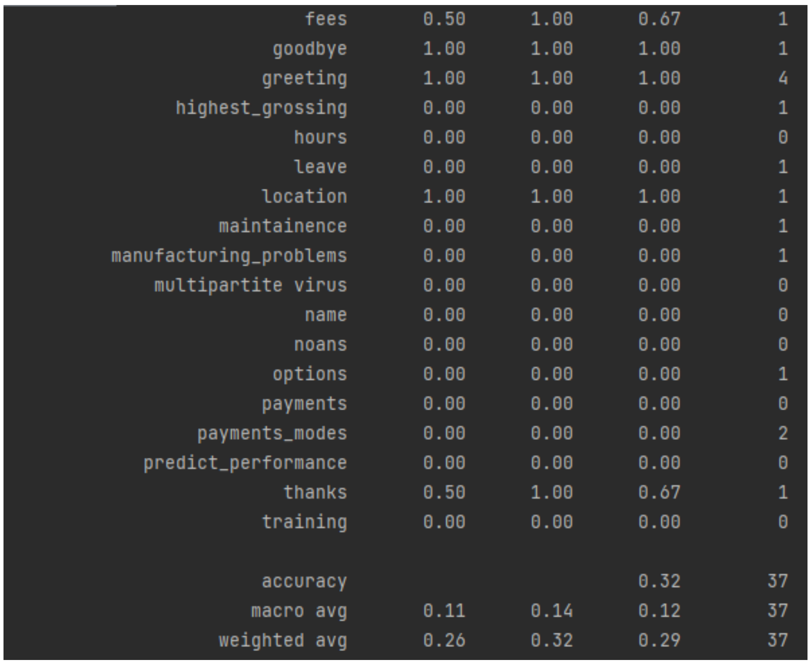
The two functions on_enter_pressed and insert message is integrated with the model code where in the model.py file will be executed and post running that the chatbot.py file will be run which will pop up the window and the user enters the message from the message entry box on the submission of which it then goes to the model through the insert message function , the bot will read the txt message inputted recognize the intent and on the basis of the intent recognized it will give the associated response back to the user.

RESULTS

Performance metrics of Distil Bert:

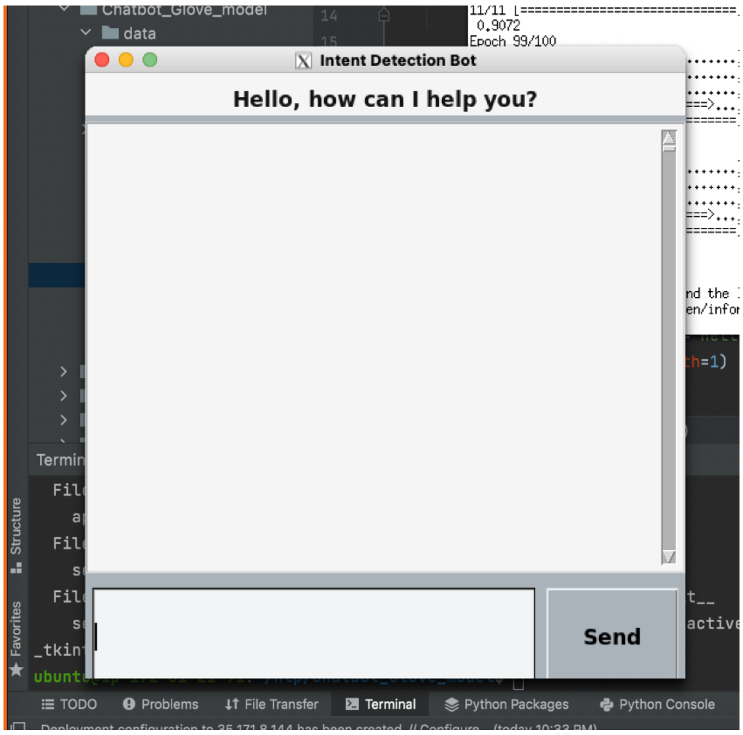


```
                                   precision    recall  f1-score   support

               file infector virus      0.00      0.00      0.00         1
                               ARP      0.00      0.00      0.00         1
          Cancellation_of_admission      0.00      0.00      0.00         1
           Certificate_Extracredit      0.00      0.00      0.00         1
                     Documents_OBC      0.00      0.00      0.00         0
                      Documents_SC      0.00      0.00      0.00         1
              Eligibility_criteria      0.00      0.00      0.00         0
                          Firewall      0.00      0.00      0.00         0
                            Gossip      0.60      1.00      0.75         3
                  Grey hat hackers      0.00      0.00      0.00         1
                HR_related_problem      0.00      0.00      0.00         1
                           Hashing      0.00      0.00      0.00         0
                             Jokes      0.00      0.00      0.00         1
                               VPN      0.00      0.00      0.00         1
                           Weather      0.00      0.00      0.00         1
                 White hat hackers      0.00      0.00      0.00         0
     admission_computerengineering      0.00      0.00      0.00         0
  admission_electronicalengineering      0.00      0.00      0.00         1
           admission_itengineering      0.00      0.00      0.00         1
```

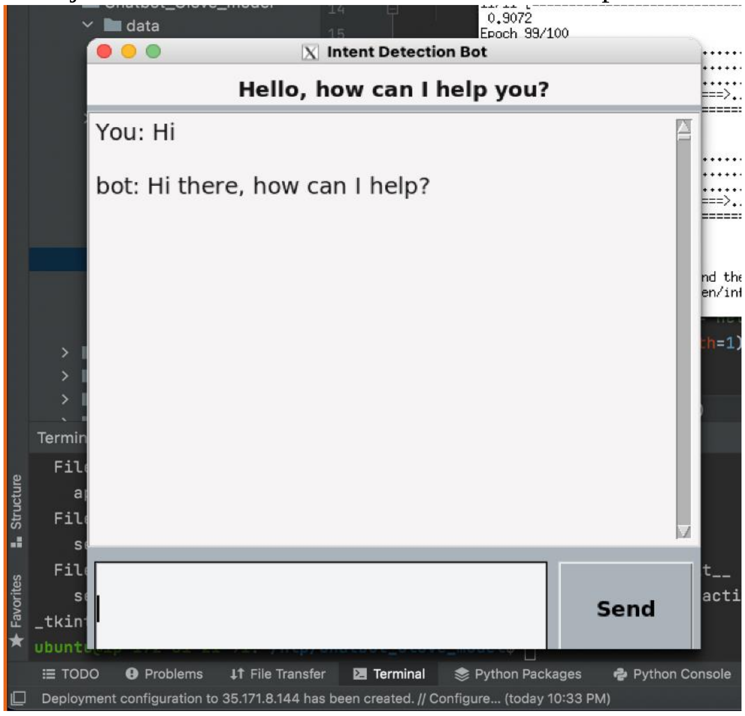


```
                              fees      0.50      1.00      0.67         1
                           goodbye      1.00      1.00      1.00         1
                          greeting      1.00      1.00      1.00         4
                  highest_grossing      0.00      0.00      0.00         1
                             hours      0.00      0.00      0.00         0
                             leave      0.00      0.00      0.00         1
                          location      1.00      1.00      1.00         1
                      maintainence      0.00      0.00      0.00         1
              manufacturing_problems      0.00      0.00      0.00         1
                 multipartite virus      0.00      0.00      0.00         0
                              name      0.00      0.00      0.00         0
                             noans      0.00      0.00      0.00         0
                           options      0.00      0.00      0.00         1
                          payments      0.00      0.00      0.00         0
                    payments_modes      0.00      0.00      0.00         2
               predict_performance      0.00      0.00      0.00         0
                            thanks      0.50      1.00      0.67         1
                          training      0.00      0.00      0.00         0

                          accuracy                          0.32        37
                         macro avg      0.11      0.14      0.12        37
                      weighted avg      0.26      0.32      0.29        37
```

Initial UI Window: This is the initial window of the Chat Bot which gets executed after running the main.py file and chat_bot.py file.



Step2: In this step we will be asking questions to chatbot which we have added as a pattern in the json file to see how the bot responds us based on the intent detected.

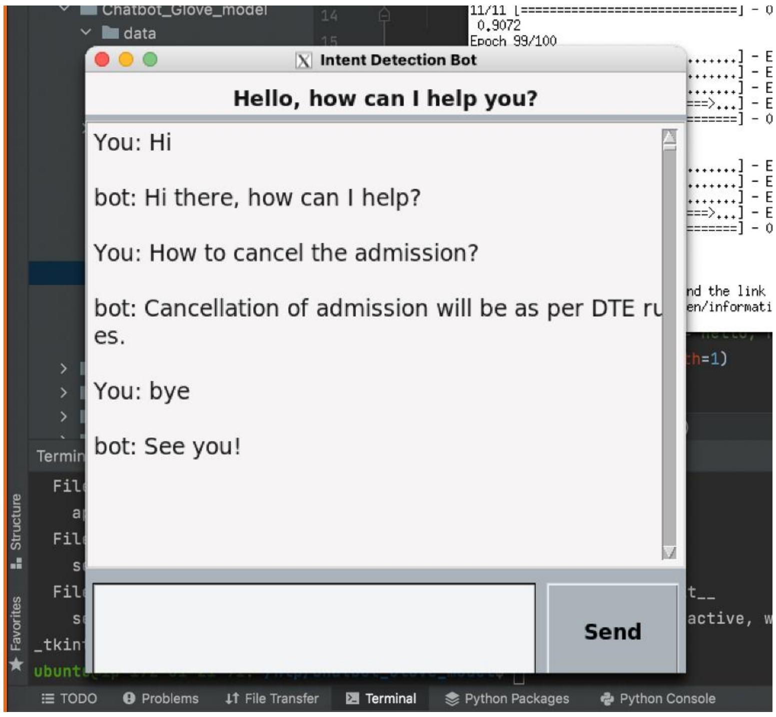


You: Hi

bot: Hi there, how can I help?

Deployment configuration to 35.171.8.144 has been created. // Configure... (today 10:33 PM)

Step3: In this window we are clearly able to see that the bot did responds us on the basis of the intent so our question to the bot was how to cancel the admission where in the intent for this question admission cancellation on the basis of the responses feed to the model was the bot gave us the exactly expected Ans to our question.



```
Intent Detection Bot
Hello, how can I help you?

You: Hi

bot: Hi there, how can I help?

You: How to cancel the admission?

bot: Cancellation of admission will be as per DTE ru
es.

You: bye

bot: See you!
```

MODEL SUMMARY

Compared to the accuracy of LSTM, Distil Bert is 0.32 which is very less. DISTIL BERT was poor since both the model is highly computational and are used for complex language use cases, the understanding we gained through the research is both the models do perform well on the training and test data of minimum 8000 entries respectively which was the major drawback of our use case since it didn’t had test set independently and was populated using train set itself. The study does say that DISTIL BERT’s performance is always considered to be degraded by 3% in comparison to BERT transformer but for our use case it did perform well with the accuracy score of 0.5% which for bert is 0.08%.

CONCLUSION

Comparing the results of other models, Distil Bert is not performing well. So, we are going ahead with LSTM model.

The further enhancements which can be done to this data is to increase the size of the train set and populate a test data. Also while doing the research for our use case we have come across ConVert which is most popularly used to deal such multi class text classification data which is related to the user queries the explanation of the model is ConveRT is a dual sentence encoder, it is effective, affordable, and quick to train also the size of the ConveRT model is less compared to the BERT model. ConveRT as per the company PolyAI who developed it was trained on Reddit conversational data (context, response). Since ConveRT has no implementation in Tensorflow hence it needs to be implemented from scratch and this will be the stage 2 of the project.

CODE PERCENTAGE

My code part contains 200 lines of code inclusive of GUI and Distil Bert models. Out of all the codes, for distil bert model has codes referred from Hugging face github. And UI was influenced from tkinter documentation.

Code percentage = 100-80 / (100+100) *100 = 10%

REFERENCES

- https://huggingface.co/docs/transformers/model_doc/distilbert
- https://arxiv.org/abs/1805.10190
- https://github.com/sonos/nlu-benchmark/tree/master/2017-06-custom-intent-engines
- https://github.com/huggingface/transformers
- https://paperswithcode.com/task/intent-detection
- https://www.sciencedirect.com/science/article/pii/S1877050918320374
- https://analyticsindiamag.com/hands-on-guide-to-word-embeddings-using-glove/