Borot

Chat Bot Project

# System Description

## Borot

**Borot** is a Chat bot project that it is designed to accept raw string query and return user’s intent and relevant info that is queried from static database.

Borot is built based on 2 NLP/NLU ideas: **Name-Entity-Recognition NER** and **Intent-Classification IC**. Also, Borot is built upon **Python Flask server** to host Machine Learning models. Briefly, when users activates Borot by typing “command”, the raw “command” is feed into NER to detect proper entities (Location, Person, etc.) and to feed into IC to detect intents. The detected entities are used to search for relevant information in Database. The final output to the user is the combination of intent, proper entities, and queried relevant info. Also, Borot is designed as a server that accept JSON-format data for GET/POST requests for Questions-Answering. The Borot’s architecture is available in Figure 2.

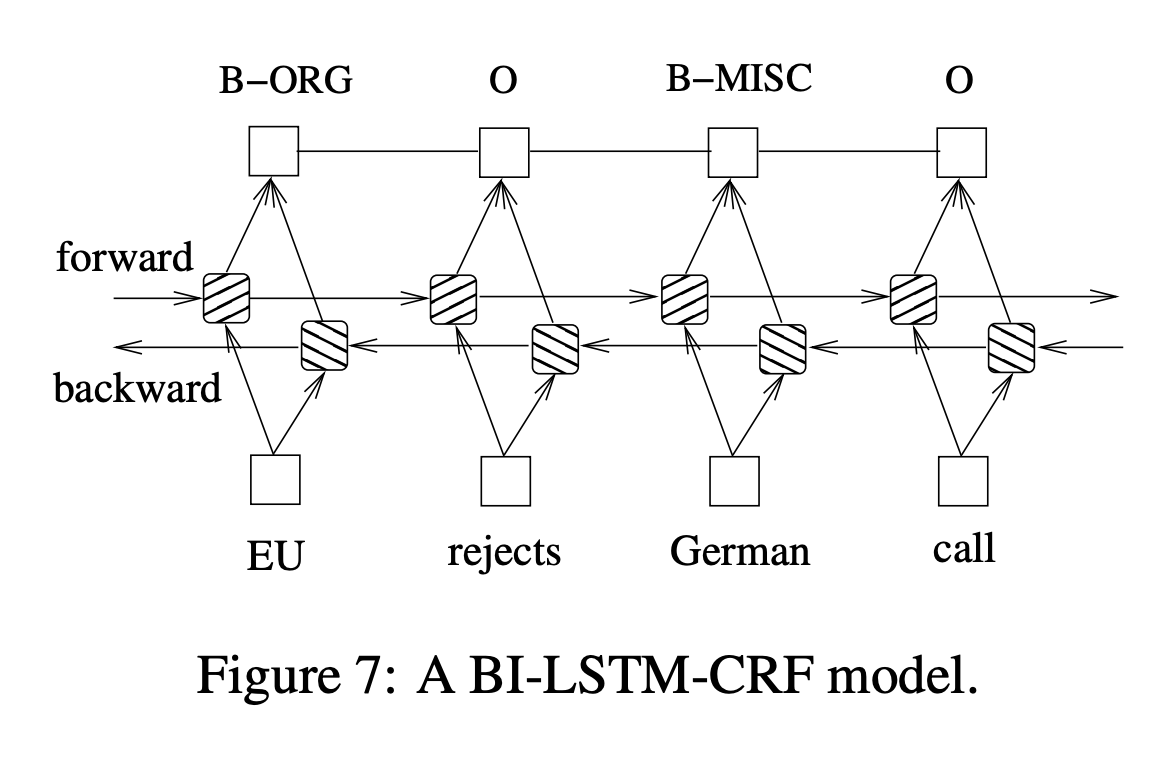


Figure 1 BiLSTM-CRF for NER

# Question & Answering Approach

The design goal of Borot is to accept query (question) and return answers (response). Hence, I attempted to implement as much as Machine Learning to help Borot to answer questions naturally. Hence, the Borot contains 3 core modules:

* Name Entity Recognizer NER
* Intent Classification IC
* Knowledge Base

## Name Entity Recognition NER

Name Entity Recognition is a NLP task to determine the proper entities (Location, Person, Time, etc.). Instead of using Dependency Parse, NER is implemented as a Deep Learning model (exactly name BiLSTM-CRF) following the research paper [1]. The model utilizes Bidirectional recurrent networks to extract Word Embeddings and classify each token for its potential entity. In the model, instead of TF-IDF (based on stats), Word Embeddings represent text as a vector for each token that it is optimized through training. The BiLSTM-CRF architecture is in Figure 1. The training and inference process is implemented in a separate Github repo that I implemented before Borot that is available [here](https://github.com/quocdat32461997/BiLSTM_CRF).

## Intent Classification IC

Intent Classification is a NLP task to determine the user’s intent in a sentence/conversation. I solved this task by training a standard Support-Vector-Machine classifier upon the dataset [2]. In [2], there are 150 predefined intents (fraud\_report, book\_hotel, book\_flight). This model is trained based on Scikit-Learn with training-accuracy at 97% and validation & test accuracies around 80%. Overfitting exists here; which means that there is likely the model makes incorrect predictions.

## Knowledge Base

Knowledge base is implemented based on Term-Frequency and Inverse-Document Frequency (TF-IDF). TF-IDF is originally designed to select important keywords based on its popularity in the document and its sparsity across corpus. The 10 documents scrapped from CNET are stored and used to compute their TF-IDFs. Given TF-IDFs, they are used to extract documents consisting detected entities by sorting TF-IDF values. The knowledge base is stored into a Pandas DataFrame in order to reduce the project complexity that the reviewers may face difficulties in setting up Borot.

# References

[1] Bidirectional LSTM-CRF for Sequence Tagging, Huang et al. <https://arxiv.org/pdf/1508.01991v1.pdf>

[2] Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. 2019. An evaluation dataset for intent classification and out-of-scope prediction. In Proceedings of EMNLP-IJCNLP.

# Instructions

* Download NER model weight following this [link](https://drive.google.com/file/d/14jZElk4UlQp4u3-EHRX2eJ4uBUYgye3n/view?usp=sharing)
* Download [Postman](https://www.postman.com/) to make REQUEST to Borot after the last step.
* Unzip the weight model and put in the folder **borot/lib/borot\_ai/BiLSTM\_CRF**
* Go to folder having file **app.py.** Then, install dependencies by typing **pip3 install -re requirements.txt**
* Type **python3 app.py** to run the Borot

# Sample Interactions

* Initialize the new user by making POST request to [**http://127.0.0.1:5000/user**](http://127.0.0.1:5000/user)

with given JSON-format data:

REQUEST:

{

"first\_name" : "Donal",

"last\_name" : "Trump",

"email" : "president@us.com"

}

RESPONSE:

"Welcome to Borot"

* To place questions/command to Borot, making POST request to [**http://127.0.0.1:5000/ask**](http://127.0.0.1:5000/ask)with given JSON-format data

REQUEST:

{

"query" : "How are you"

}

RESPONSE:

{

"response": {

"Entities": {

"geo": [

"who"

],

"nat": [

"am"

],

"per": [

"i"

]

},

"Intent": "pto\_request\_status",

"Relevant\_info": null

},

"status\_code": 200

}

* The answer from Borot is in JSON format including:
  + Entities: Detected entities such as Geo (Location), Nat (country), and Per (proper name)
  + Intent: one of 150 predefined intents
  + Relevant info: either Null or text if found.

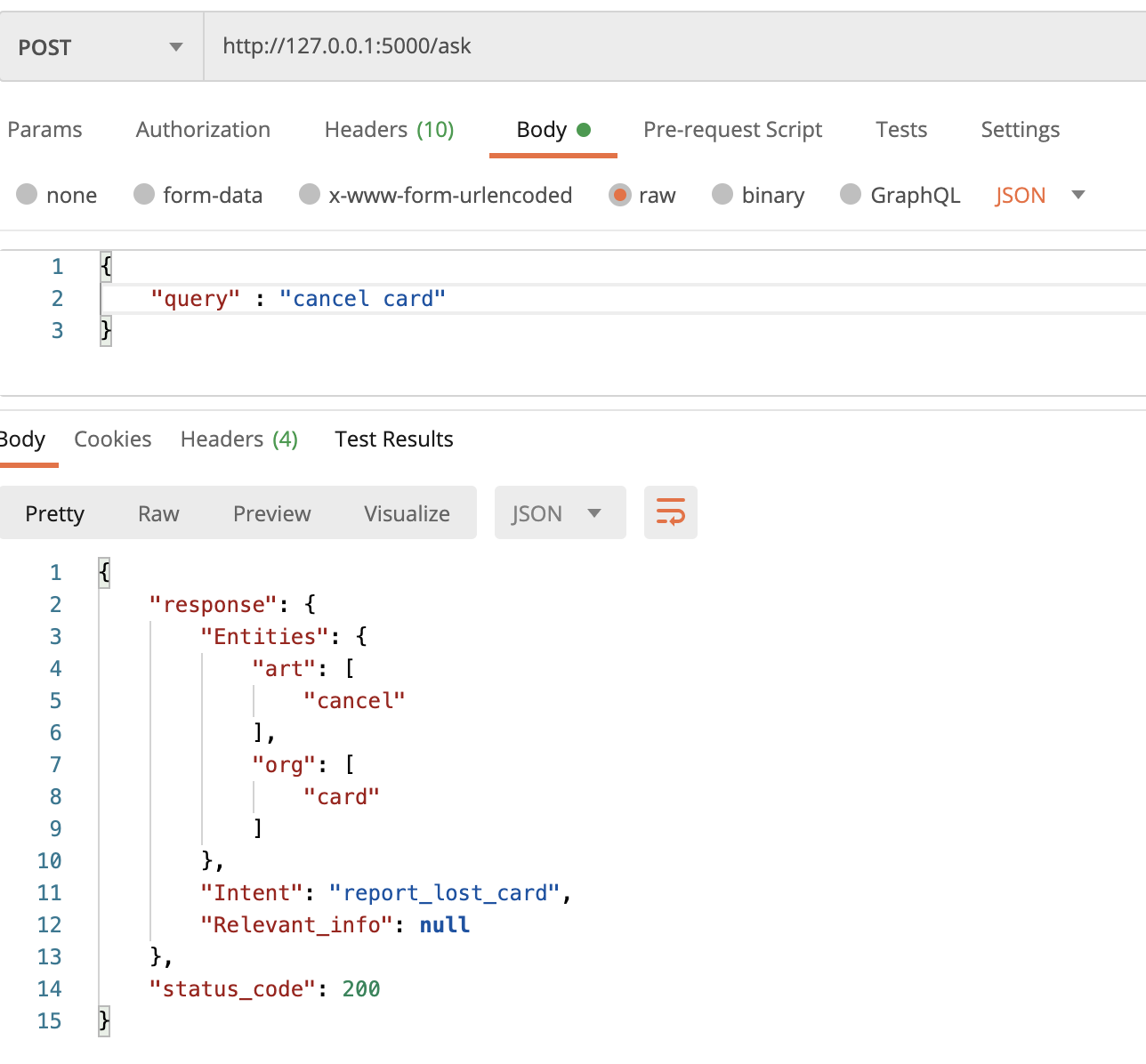


Figure 2 Interaction of Borot to query "cancel card"

# Evaluation

The chatbot I chose to implement is limited in terms of generating natural text response given questions. This is the challenging NLP problem. In order to improve the chatbot intelligence, it is a need to collect more data and perform training Machine Learning algorithms to help the chatbot detect intents better. Also, it is necessary to frequently update the knowledge base of the chatbot. Because the knowledge base serves as the answer of the chat bot the user.

## Comprehensive capabilities

The model is capable of detecting 150 intents and classifying 15 Name Entities thanks to NER and IC modules. These 2 modules are hosted within local server that could quickly perform tasks. The limit of the chatbot is when users make out-of-scope questions; that means users’ questions are not seen by the 2 NER and IC modules before. If that is the case, the chatbot may make irrelevant answers.

## Complexity

Borot is a complicated system that is embedded multiple Github repos that I created in the past. Also, the past Github repos I created are mostly for training Machine Learning with good validation performance. However, by observing the chatbot interactions, it performs worse. This could be explained due to the limited dataset when training Machine Learning.

## Speed

Despite a complex system, Borot is almost similar to current Chatbot systems (I believe) in terms of structure. NER and IC are 2 must-have steps in any chatbot systems. Despite the complexity, the model still response fast.

## User-Computer Interaction

As I designed Borot the chatbot as a local server, it requires users to perform RESTful requests that it is not user-friendly. It requires users to understand and know how to make REQUESTS to the server.

# Dependencies

* tensorflow==2.3.1
* numpy==1.18.5
* Flask==1.1.2
* pandas==1.1.3
* boto3==1.16.6
* tensorflow\_addons==0.11.2
* nltk==3.5
* mysql\_connector\_repackaged==0.3.1
* scikit\_learn==0.23.2

# Predefined Intent list

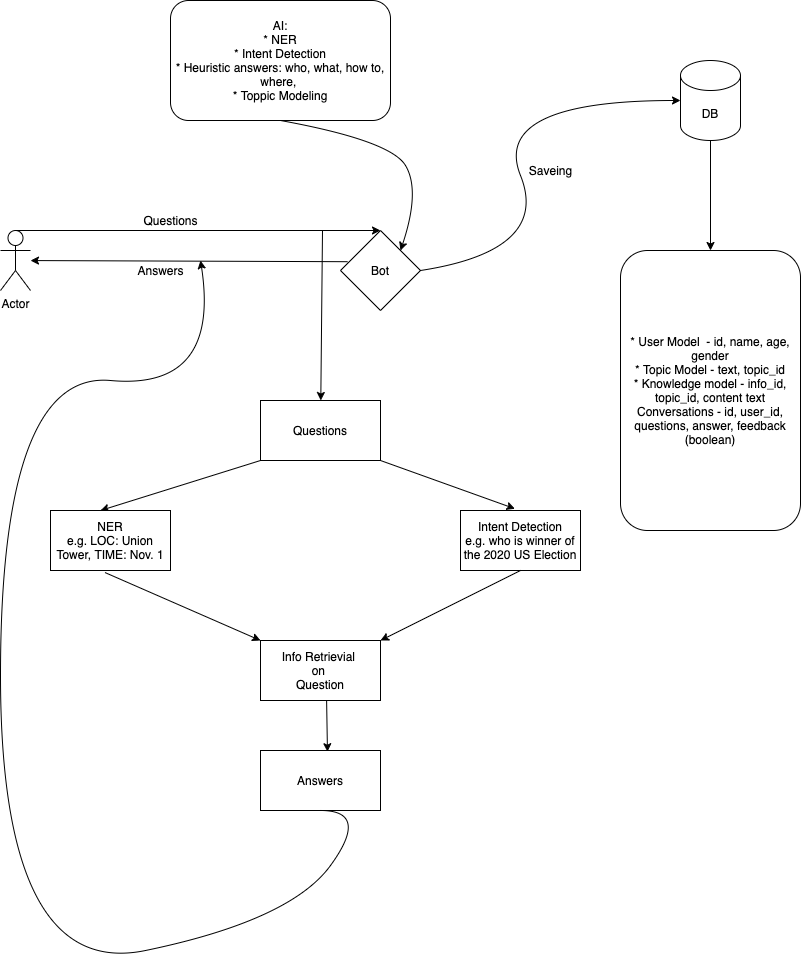


Figure 3 Architecture of Borot

1. play\_music
2. next\_song
3. application\_status
4. plug\_type
5. gas
6. order
7. credit\_limit\_change
8. directions
9. meaning\_of\_life
10. greeting
11. calculator
12. change\_ai\_name
13. insurance
14. reset\_settings
15. weather
16. oil\_change\_how
17. date
18. pto\_balance
19. reminder
20. todo\_list
21. order\_checks
22. yes
23. do\_you\_have\_pets
24. spending\_history
25. what\_can\_i\_ask\_you
26. maybe
27. rewards\_balance
28. schedule\_meeting
29. recipe
30. goodbye
31. cancel
32. mpg
33. report\_lost\_card
34. restaurant\_suggestion
35. tell\_joke
36. bill\_due
37. book\_flight
38. ingredients\_list
39. change\_accent
40. interest\_rate
41. book\_hotel
42. last\_maintenance
43. what\_is\_your\_name
44. international\_visa
45. timer
46. expiration\_date
47. direct\_deposit
48. car\_rental
49. card\_declined
50. pin\_change
51. account\_blocked
52. user\_name
53. definition
54. tire\_pressure
55. calendar\_update
56. smart\_home
57. income
58. change\_language
59. cook\_time
60. accept\_reservations
61. restaurant\_reviews
62. meal\_suggestion
63. w2
64. no
65. alarm
66. what\_song
67. todo\_list\_update
68. transfer
69. thank\_you
70. international\_fees
71. measurement\_conversion
72. freeze\_account
73. spelling
74. new\_card
75. transactions
76. pto\_used
77. text
78. tire\_change
79. find\_phone
80. balance
81. replacement\_card\_duration
82. change\_volume
83. where\_are\_you\_from
84. food\_last
85. ingredient\_substitution
86. next\_holiday
87. change\_speed
88. what\_are\_your\_hobbies
89. exchange\_rate
90. timezone
91. apr
92. translate
93. uber
94. reminder\_update
95. traffic
96. travel\_notification
97. damaged\_card
98. bill\_balance
99. flight\_status
100. whisper\_mode
101. gas\_type
102. fun\_fact
103. improve\_credit\_score
104. routing
105. time
106. travel\_alert
107. pay\_bill
108. taxes
109. meeting\_schedule
110. order\_status
111. payday
112. sync\_device
113. oos
114. schedule\_maintenance
115. lost\_luggage
116. update\_playlist
117. calories
118. distance
119. roll\_dice
120. nutrition\_info
121. change\_user\_name
122. cancel\_reservation
123. pto\_request
124. jump\_start
125. travel\_suggestion
126. share\_location
127. are\_you\_a\_bot
128. rollover\_401k
129. redeem\_rewards
130. shopping\_list\_update
131. who\_made\_you
132. who\_do\_you\_work\_for
133. min\_payment
134. credit\_limit
135. how\_busy
136. report\_fraud
137. carry\_on
138. insurance\_change
139. pto\_request\_status
140. how\_old\_are\_you
141. confirm\_reservation
142. shopping\_list
143. repeat
144. flip\_coin
145. calendar
146. restaurant\_reservation
147. oil\_change\_when
148. vaccines
149. current\_location
150. credit\_score
151. make\_call

Borot is tested and run well on MacOS system. If you encounter any difficulty on running Borot. Please contract me.