Natural Language Processing

Shopping Assistant Chatbot

Timo Achtelik, 751364

Introduction

As companies try to ensure that their customers can access the right information anytime, anywhere, it now seems inevitable to integrate chatbots into business platforms or websites. A conversation chatbot is an AI-driven intelligent software that enables machines to understand, process and respond to human speech based on complex deep learning and natural language understanding(NLU).

The following chatbot was programmed and trained with Python, Tensorflow and tflearn. JSON files with tags, patterns and corresponding answers were used as data for the training.

Requirements

The chatbot should include the following features:

- a detection of keywords and/or key phrases, with appropriate responses
- an „intelligent“ continuation of the dialogue in case no keyword or key phrase was detected
- a random generation & selection of answers
- a „you – I reversal“ in the dialogue (e.g. Input: I am happy. Answer: Why are you happy?)
- at least one „intelligent component“, such as remembering and referring to what has been said before, i.e. prior to the directly preceding input.

Data

A chatbot framework needs a structure in which the conversation intentions are defined. The scope of the chatbot depends on the amount of data used for training. The data set used in this project is a combination of examples and common patterns for an event search. The corresponding responses were filled with real events, data and prices from Eventim.

The data is structured in tags, patterns, responses and context.
Tags: Possible classes of user intentions for asking a question.
Patterns: The way users would normally ask questions if they wanted to have a ticket for an event
Responses: Predefined responses for each tag in the data set, from which a response is randomly selected

```
{"intents": [
        {"tag": "start_conversation",
         "patterns": ["hi there", "is anyone there?","hey","hola", "hello", "good day","hi"],
         "responses": ["Hello there! I am Eventify and i suggest events for you :) You can either ask me for
        },
        {"tag": "end_conversation",
         "patterns": ["bye", "see you later", "goodbye", "nice chatting to you, bye", "till next time"],
         "responses": ["Have a lovely day!", "Bye :)", "Have a nice time at your next event", "Enjoy the ever
        },
        {"tag": "thanks",
         "patterns": ["thanks", "thank you", "you're the best", "awesome, thanks", "thanks for helping me"],
         "responses": ["Happy to help!", "Any time!", "My pleasure", "Always :)", "No thank you, for believi
        },
        {"tag": "noanswer",
         "patterns": [],
         "responses": ["Sorry, kindly rephrase the question","Sorry, can't understand you", "Please give me r
        },
        {"tag": "options",
         "patterns": ["how can you help me?", "what can you do?", "what help you provide?", "how can you be h
         "responses": ["I can help you find your next event", "I can suggest events for you", "Tell me what y
        },
```


Training

Text pre-processing

Before the model is trained with the data, it must first be prepared. This includes for example the division of sentences into individual words or the handling of punctuation. The cleansing of text data in NLP is task-specific. For the chatbot, the following tasks, among others, were performed:

- Tokenization
- Stemming of words
- Removing stop words

Tokenization is the process of splitting a text corpus into individual words, i.e., splitting phrases, sentences, paragraphs, or entire text documents into smaller units, such as individual words or terms. Each of these smaller units is called a token. Tokenization can be done manually by splitting based on spaces or using special tools in libraries like NLTK. The function below was used to tokenize the body:

```python
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = nltk.word_tokenize(pattern)
        # add to our words list
        words.extend(w)
        # add to documents in our corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Subsequently, the words were stemmed and stop words were removed. Stop words are words that do not contribute to the deeper meaning of the sentence - definite and indefinite articles, pronouns and conjunctions, to name a few. The truncation of words has the advantage that words with the same stem match. For example, the root 'tak' will match 'take', 'taking', 'takers', etc.

```
        # stem and lower each word and remove duplicates
        words = [stemmer.stem(w.lower()) for w in words if w not in stop_words]
        words = sorted(list(set(words)))
        # remove duplicates
        classes = sorted(list(set(classes)))
```

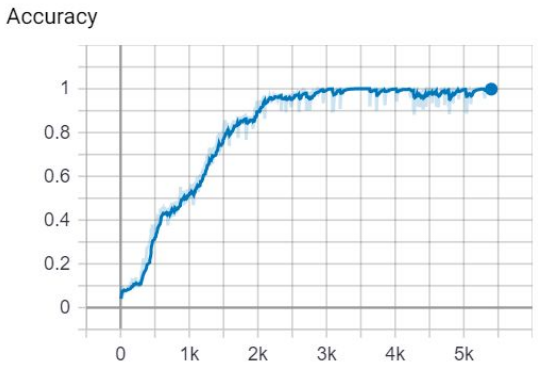
```
108 documents
26 classes ['basket', 'checkout', 'comedy', 'comedy_darmstadt',
'comedy_frankfurt', 'concert', 'concert_frankfurt', 'dinner_krimi',
'ehrlich_brothers', 'end_conversation', 'events_darmstadt', 'events_frankfurt',
'events_general', 'future_palace', 'glanz_auf_dem_vulkan', 'goetz_widmann',
'musical', 'musical_darmstadt', 'new_york', 'options', 'party', 'party_frankfurt',
'phrases', 'start_conversation', 'thanks', 'times']
84 unique stemmed words ["'re", ',', '-', '20er', ':', '?', 'ad', 'anyon',
 'auf', 'awesom', 'basket', 'best', 'blau', 'broth', 'buy', 'bye', 'categ',
 'chat', 'checkout', 'comedy', 'concert', 'conert', 'darmstadt', 'das', 'day',
 'dem', 'die', 'din', 'dinnerkrim', 'ehrlich', 'ein', 'escap', 'ev', 'find',
 'frankfurt', 'fut', 'glanz', 'go', 'goetz', 'good', 'goodby', 'gospel', 'hello',
 'help', 'hey', 'hi', 'hol', 'how', 'in', 'jahr', 'kind', 'know', 'krim', 'lat',
 'let', 'much', 'mus', 'new', 'next', 'nic', 'off', 'palac', 'party', 'pay',
 'pleas', 'provid', 'ready', 'schuss', 'see', 'show', 'snow', 'star',  'support',
 'thank', 'ticket', 'til', 'tim', 'today', 'tomorrow', 'tour', 'vulk', 'want', 'widman', 'york']
```

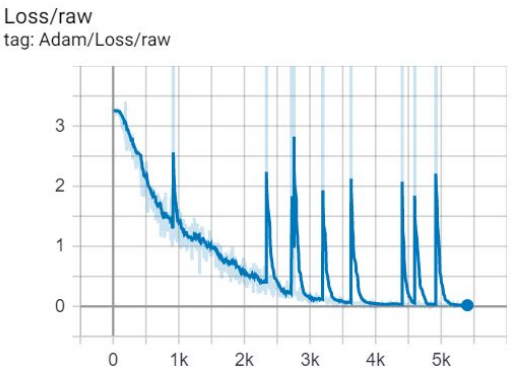
In the following, the model was trained. The first 5 lines define the neural 'net' with a sequence of Tflearn functions: from tflearn.input_data over tflearn.fully_connected to tflearn.regression. The input data has 5 characteristics, 32 nodes are used in each hidden layer and the output has 2 classes.
After that, the deep neural network is initiated: tflearn.DNN with the network, and a tensorboard parameter to activate logging.
Finally, the model is adapted to the data and trained.

```
# Build neural network
net = tflearn.input_data(shape=[None, len(train_x[0])])
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, len(train_y[0]), activation='softmax')
net = tflearn.regression(net)

# Define model and setup tensorboard
model = tflearn.DNN(net, tensorboard_dir='tflearn_logs')
# Start training (apply gradient descent algorithm)
model.fit(train_x, train_y, n_epoch=600, batch_size=12, show_metric=True)
model.save('model.tflearn')
```

The accuracy and loss of the model are shown graphically below. It can be observed that the model has an accuracy of 1 after only 3000 steps.





Intelligent component

For the integration of an intelligent component, a shopping cart was implemented. The user can add one ticket at a time and the total costs are calculated in the next step. To do this, the user must first request information about the event. In this case, the price of the event is automatically saved in a help list. If the user enters a sentence like "add to basket" or "add my tickets to basket", the price of the ticket is added to the basket. This is only possible if the input follows the information of the event. If the user enters something else, the price will be removed from the help list.
It is possible to add several events to the shopping cart and have the total price calculated if the input matches the "checkout" tag.

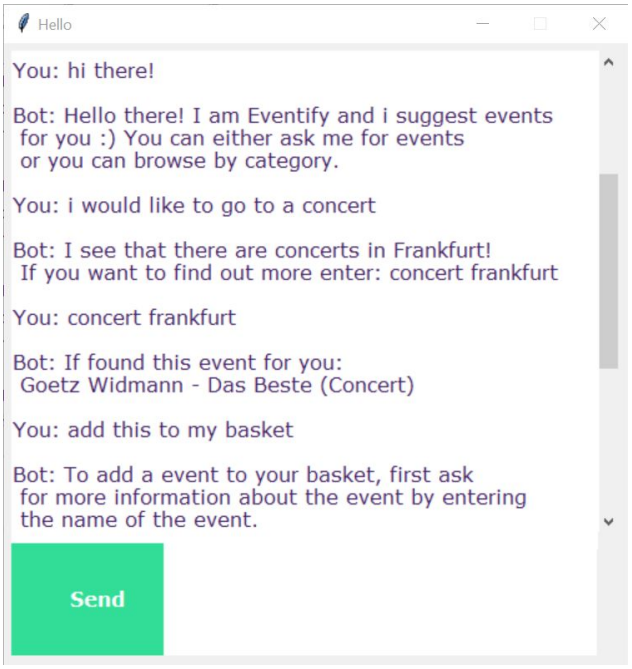
```
if i['tag'] == 'basket' and addBasket:
    basket.extend(helper)
    helper.clear()
else:
    addBasket=False
    helper.clear()
for y in events:
    if i['tag'] == y[0]:
        helper.append(y[1])
        addBasket=True
if i['tag'] == 'checkout':
    total = 0
    for li in basket:
        total = total + li
    sentence = "You have to pay a total of {} Euro"
    return sentence.format(total)
```

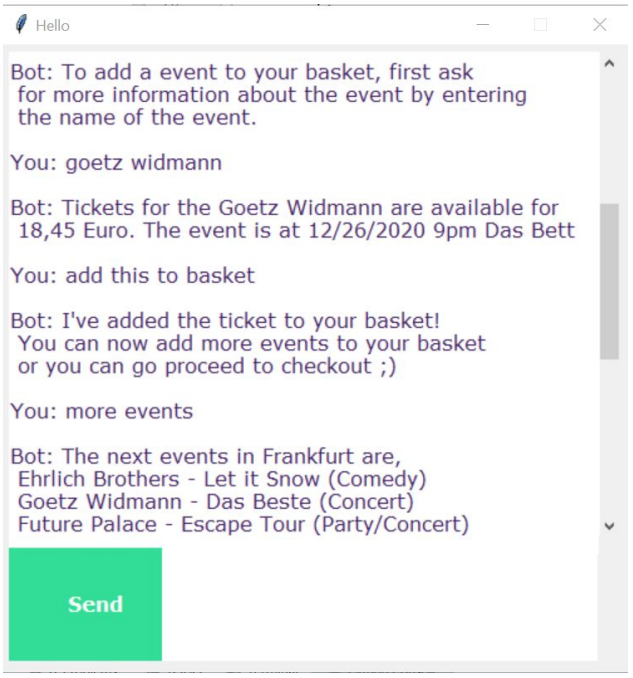
Interaction

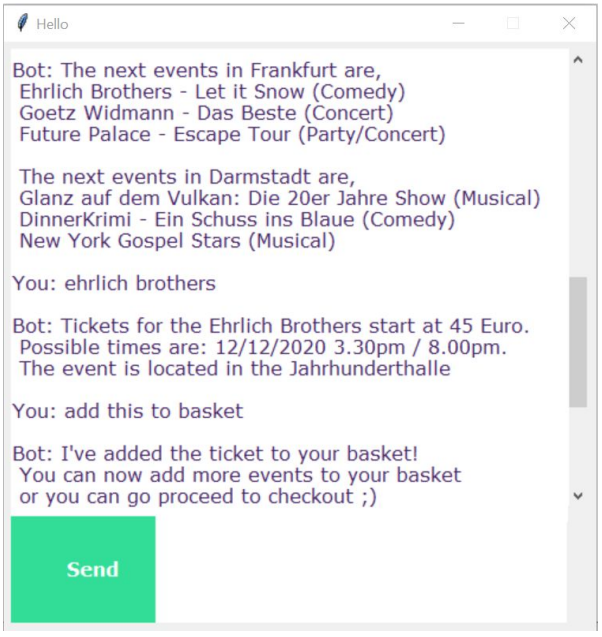
To process the input, the sentences must be processed and separated into their individual components (words), just like when learning the model. Afterwards, the input is classified and assigned to a tag with a probability. Afterwards, an answer is randomly selected from the possible answers for this tag and given to the user. Each sentence from the input passed to response() is classified. The classifier uses model.predict () and is very fast. The probability returned by the model is used with the intent definition to create a list of possible responses. If one or more classifications are above the threshold, it can be determined whether the tag matches the intention and processed accordingly. The classification list is considered a stack and jumps from the stack to find matches until a match is found or the list is empty.

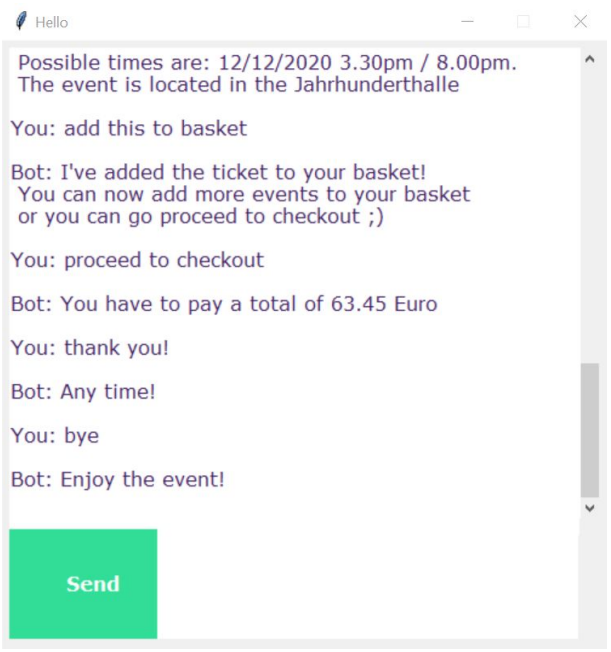
```
ERROR_THRESHOLD = 0.25
def classify(sentence):
    # generate probabilities from the model
    results = model.predict([bow(sentence, words)])[0]
    # filter out predictions below a threshold
    results = [[i,r] for i,r in enumerate(results) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append((classes[r[0]], r[1]))
    # return tuple of intent and probability
    return return_list
```

For the interaction with the chatbot, a GUI with tkinter was built.









Conclusion

The chatbot fulfills the above-mentioned requirements and can interpret the user input with quite a high accuracy. With more data and more time, it would also be possible to interact with the chatbot even better and more. Currently, it only fulfills simple basic functions, but in an updated version it could be possible to add more tickets to the shopping cart or to have the bot interpret the user input even better.