Project Report

----NLP Research Project

Date: 2019-08-10

Contents

1.Backgroud

Natural language processing (NLP) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

The history of natural language processing(NLP) generally started in the 1950s, although work can be found from earlier periods.

In the early days, many language-processing systems were designed by hand-coding a set of rules, e.g. by writing grammars or devising heuristic rules for stemming. However, this is rarely robust to natural language variation.

Since the so-called "statistical revolution" in the late 1980 s and mid 1990s, much natural language processing research has relied heavily on machine learning.

The machine-learning paradigm calls instead for using statistical inference to automatically learn language rules through the analysis of large corpora of typical real-world examples (a corpus (plural, "corpora") is a set of documents, possibly with human or computer annotations).

In the 2010s, representation learning and deep neural network-style machine learning methods became widespread in natural language processing, due in part to a flurry of results showing that such techniques can achieve state-of-the-art results in many natural language tasks, for example in language modeling, parsing, and many others. Popular techniques include the use of word embeddings to capture semantic properties of words, and an increase in end-to-end learning of a higher-level task (e.g., question answering) instead of relying on a pipeline of separate intermediate tasks (e.g., part-of-speech tagging and dependency parsing).

About the Project:

The project focuses on practicing natural language understanding with the python module “spacy” and “rasa_nlu”. The final aim is building a chat bot answering specific kind of questions.

2. Key Methods

2.1 Regular Expression

Description:

A regular expression is a sequence of characters that define a search pattern. Usually such patterns are used for "find" or "find and replace" operations on strings, or for input validation.

Regular expression a technique developed in theoretical computer science and formal language theory. The concept arose in the 1950s when the American mathematician Stephen Cole Kleene formalized the description of a regular language. The concept came into common use with Unix text-processing utilities. Different syntaxes for writing regular expressions have existed since the 1980s, one being the POSIX standard and another, widely used, being the Perl syntax.

The phrase regular expressions is often used to mean the specific, standard textual syntax for representing patterns for matching text. Each character in a regular expression is either a metacharacter, having a special meaning, or a regular character that has a literal meaning. For example, in the regex a., a is a literal character which matches just 'a', while '.' is a meta character that matches every character except a newline. Therefore, this regex matches, for example, 'a ', or 'ax', or 'a0'.

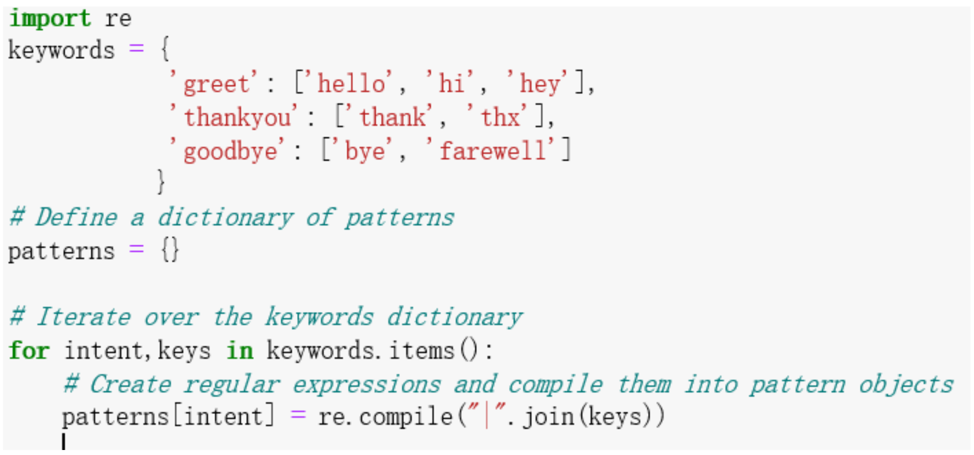
Syntax:

Character	Description	Example	Sample Match
\d	Most engines: one digit from 0 to 9	file_\d\d	file_25
\w	Most engines: "word character": ASCII letter, digit or underscore	\w-\w\w\w	A-b_1
\s	Most engines: "whitespace character": space, tab, newline, carriage return, vertical tab	a\sb\sc	a b c
\D	One character that is not a *digit* as defined by your engine's *\d*	\D\D\D	ABC
\W	One character that is not a *word character* as defined by your engine's *\w*	\W\W\W\W\W	*-+=)
\S	One character that is not a *whitespace character* as defined by your engine's *\s*	\S\S\S\S	Yoyo

Quantifier	Description	Example	Sample Match
+	One or more	Version \w-\w+	Version A-b1_1
{3}	Exactly three times	\D{3}	ABC
{2,4}	Two to four times	\d{2,4}	156
{3,}	Three or more times	\w{3,}	regex_tutorial
*	Zero or more times	A*B*C*	AAACC
?	Once or none	plurals?	plural

Usage in the project:

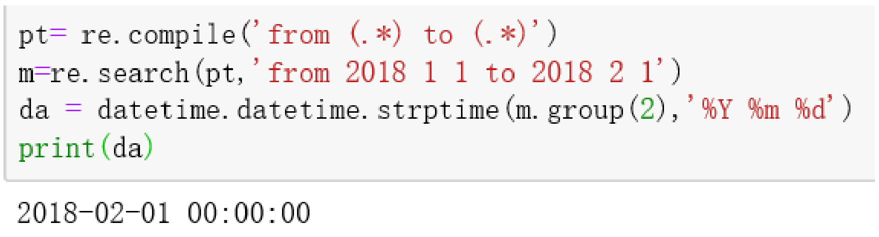
Regular expression is used in the project to match specific patterns so that a intent or entity can be extracted from a sentence.



```
import re
keywords = {
            'greet': ['hello', 'hi', 'hey'],
            'thankyou': ['thank', 'thx'],
            'goodbye': ['bye', 'farewell']
           }
# Define a dictionary of patterns
patterns = {}

# Iterate over the keywords dictionary
for intent,keys in keywords.items():
    # Create regular expressions and compile them into pattern objects
    patterns[intent] = re.compile("|".join(keys))
```

Extracting "greet" intent



```
pt= re.compile('from (.*) to (.*)')
m=re.search(pt,'from 2018 1 1 to 2018 2 1')
da = datetime.datetime.strptime(m.group(2),'%Y %m %d')
print(da)
```

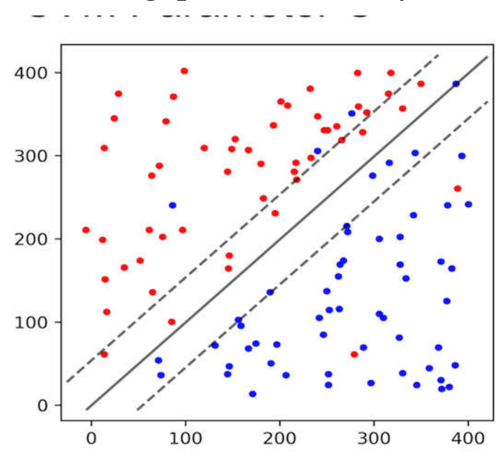
```
2018-02-01 00:00:00
```

Extracting the entity of date

2.2 Support Vector Machine

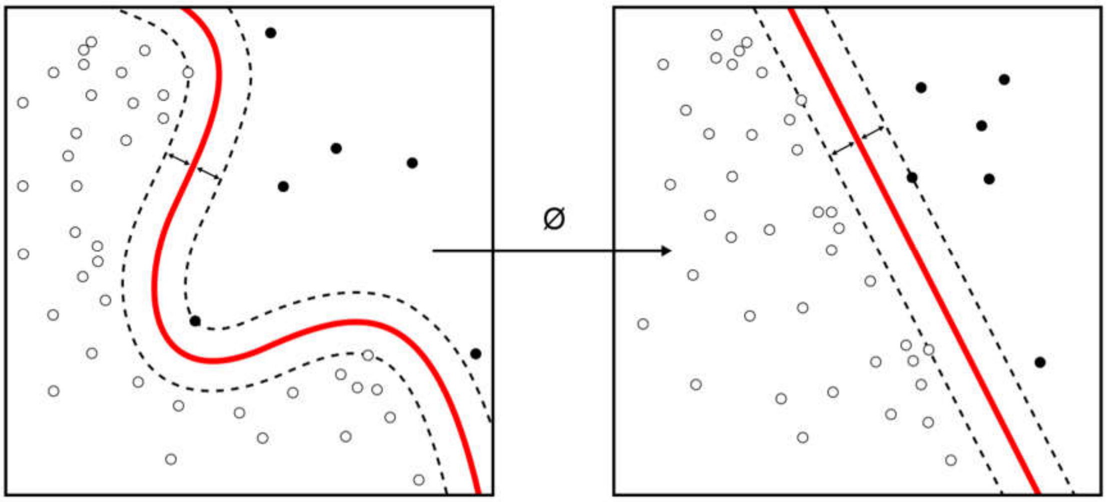
Description:

In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.



SVM linear classification

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

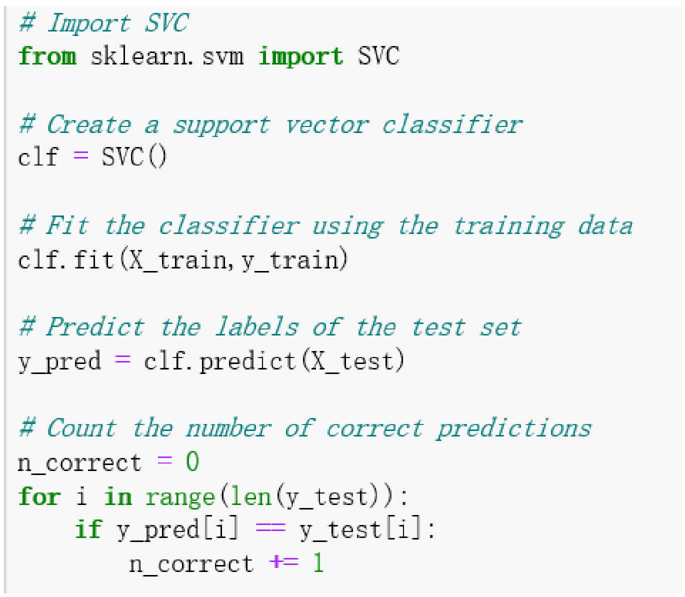


SVM non-linear classification

Usage in the project:

The project uses the SVC module from sklearn.svm to construct

a SVM for classification.



```python
# Import SVC
from sklearn.svm import SVC

# Create a support vector classifier
clf = SVC()

# Fit the classifier using the training data
clf.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Count the number of correct predictions
n_correct = 0
for i in range(len(y_test)):
    if y_pred[i] == y_test[i]:
        n_correct += 1
```

SVM classification in the project

2.3 SQL in Python

SQL examples:

SELECT *column_name,column_name*

FROM *table_name*

WHERE *column_name operator value*;

SELECT *column_name,column_name*

FROM *table_name*

ORDER BY *column_name,column_name*

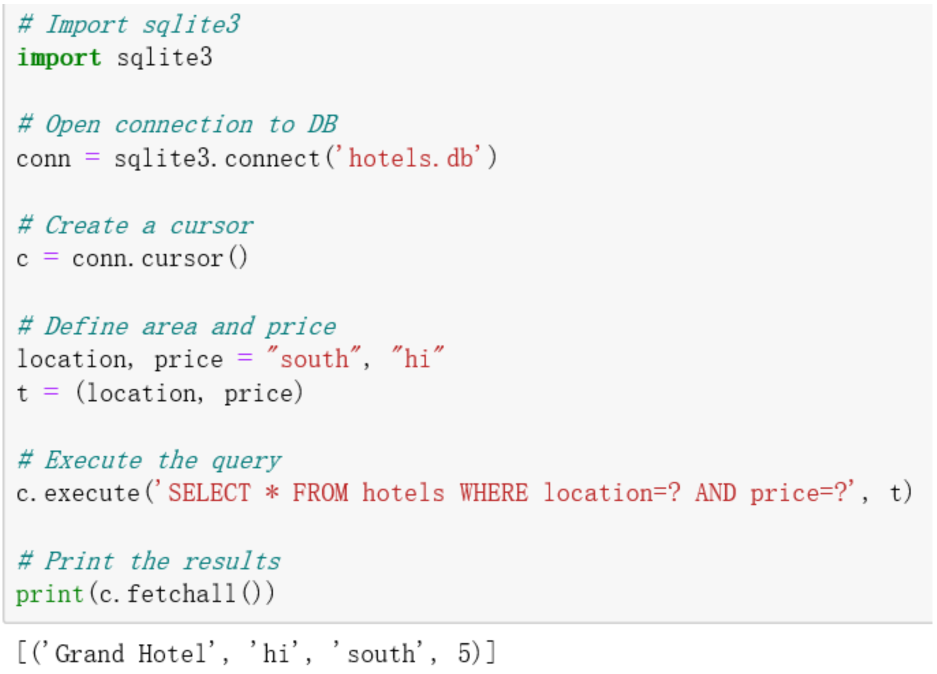
DELETE FROM *table_name*

WHERE *some_column=some_value*;

In python:

We can import the library 'sqlite3' to use SQL in python.

The following picture shows the usage of SQL in python in the project:



```
# Import sqlite3
import sqlite3

# Open connection to DB
conn = sqlite3.connect('hotels.db')

# Create a cursor
c = conn.cursor()

# Define area and price
location, price = "south", "hi"
t = (location, price)

# Execute the query
c.execute('SELECT * FROM hotels WHERE location=? AND price=?', t)

# Print the results
print(c.fetchall())
```

```
[('Grand Hotel', 'hi', 'south', 5)]
```

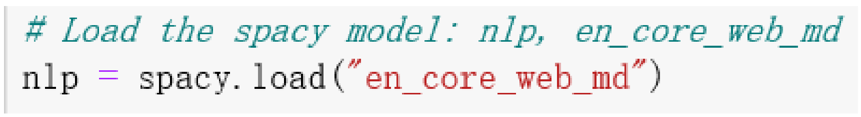
2.4 Spacy and Rasa_nlu

Spacy is an “industrial-strength” natural language processing module in python. Rasa NLU is a language processing tool for intent classification and entity extraction in chatbots.

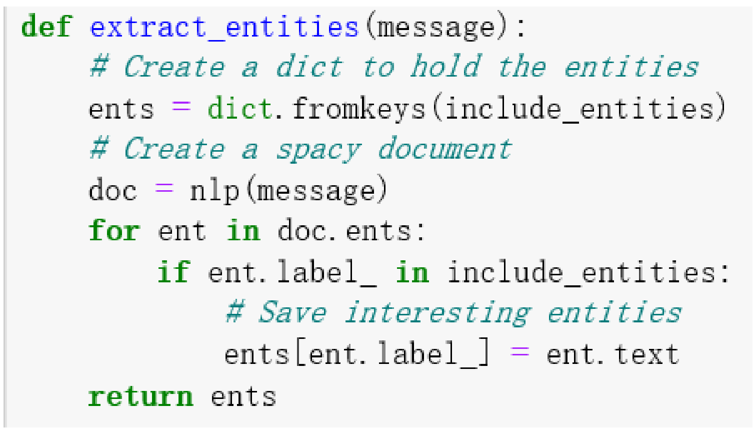
Spacy has the following features: Non-destructive tokenization, Named entity recognition, Pre-trained word vectors, Labelled dependency parsing, Syntax-driven sentence segmentation, Built in visualizers for syntax and NER, Export to numpy data arrays, Efficient binary serialization, Easy model packaging and deployment, Robust and rigorously evaluated accuracy

Usage in the project:

In the project, spacy is used for entity recognition, similarity comparison and dependency analysis and rasa_nlu is used for entity recognition and intent recognition.

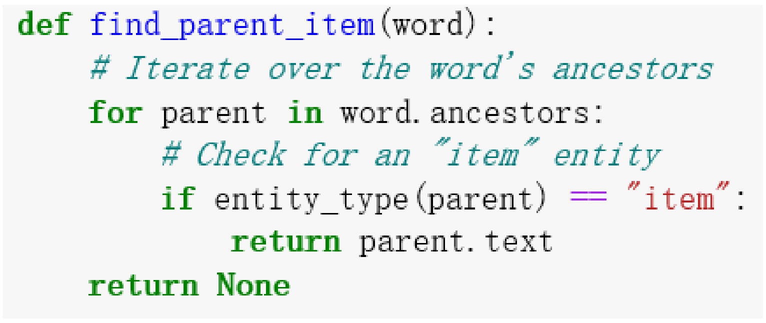


```python
# Load the spacy model: nlp, en_core_web_md
nlp = spacy.load("en_core_web_md")
```



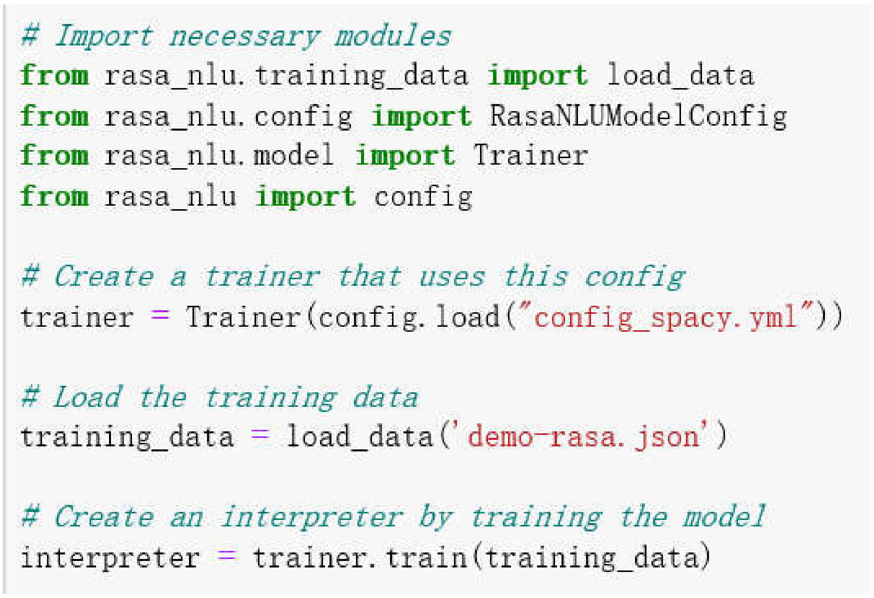
```python
def extract_entities(message):
    # Create a dict to hold the entities
    ents = dict.fromkeys(include_entities)
    # Create a spacy document
    doc = nlp(message)
    for ent in doc.ents:
        if ent.label_ in include_entities:
            # Save interesting entities
            ents[ent.label_] = ent.text
    return ents
```

Entity extraction with spacy



```python
def find_parent_item(word):
    # Iterate over the word's ancestors
    for parent in word.ancestors:
        # Check for an "item" entity
        if entity_type(parent) == "item":
            return parent.text
    return None
```

Dependency analysis with spacy



```python
# Import necessary modules
from rasa_nlu.training_data import load_data
from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer
from rasa_nlu import config

# Create a trainer that uses this config
trainer = Trainer(config.load("config_spacy.yml"))

# Load the training data
training_data = load_data('demo-rasa.json')

# Create an interpreter by training the model
interpreter = trainer.train(training_data)
```

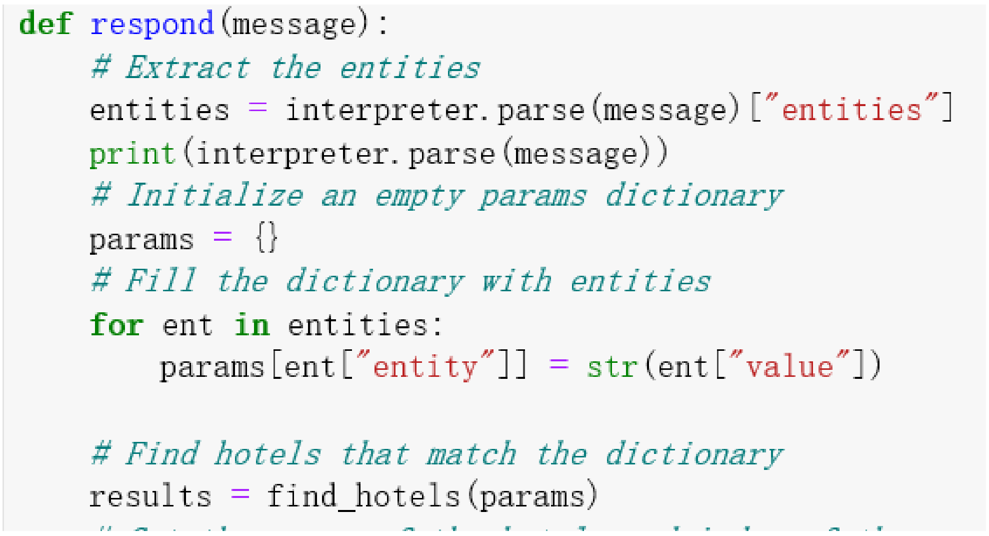
Train a model with rasa_nlu

2.5 Incremental Slot Filling and Negation

Description:

Incremental slot filling is used for multi-round dialogue bot. Each round the entities and intent in users' message are extracted and stored into a "slot". The stored dada can be used in later rounds. Negation of entities is also extracted and stored. The bot replies to users' message according to stored data.

Usage in the project:



```python
def respond(message):
    # Extract the entities
    entities = interpreter.parse(message)["entities"]
    print(interpreter.parse(message))
    # Initialize an empty params dictionary
    params = {}
    # Fill the dictionary with entities
    for ent in entities:
        params[ent["entity"]] = str(ent["value"])

    # Find hotels that match the dictionary
    results = find_hotels(params)
```

Entities are stored in the dictionary "params"

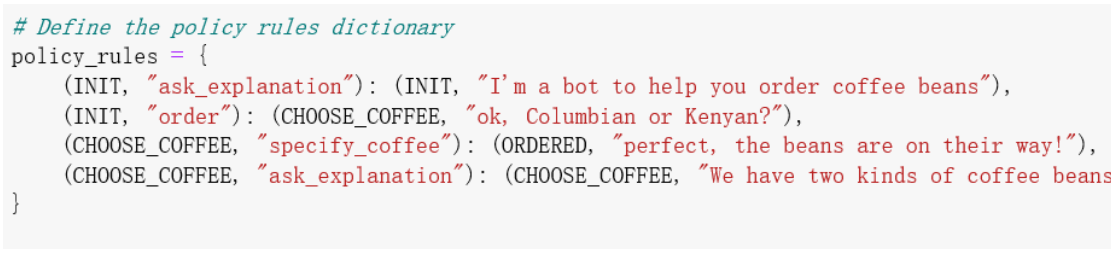
2.6 Finite state machine

Description:

A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition.

Usage in the project:

In the project, state machine is used for multi-round dialogue bot. The bot reply to users' messages differently in different states. State transition is triggered by users' messages.



```python
# Define the policy rules dictionary
policy_rules = {
    (INIT, "ask_explanation"): (INIT, "I'm a bot to help you order coffee beans"),
    (INIT, "order"): (CHOOSE_COFFEE, "ok, Columbian or Kenyan?"),
    (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their way!"),
    (CHOOSE_COFFEE, "ask_explanation"): (CHOOSE_COFFEE, "We have two kinds of coffee beans
}
```

Define transitions

3.the Final Project

Project Requirements:

The final project is a to build a chat bot on stock information. The chat bot should have the following functions: understand users' question on stock information, answer the questions and have multi-round dialogue. The bot should be deployed on Wechat.

Analysis:

Spacy and rasa_nlu can be used to extract the intent and entities of user messages. Multi-round dialogue can be achieved using incremental slot filling and state machine.

Function description:

Finally, the chat bot can achieve the following functions:

1. User log in.

2. Recognize the intent of asking the price, volume, marketcap, and historical data of a stock and answer the question.

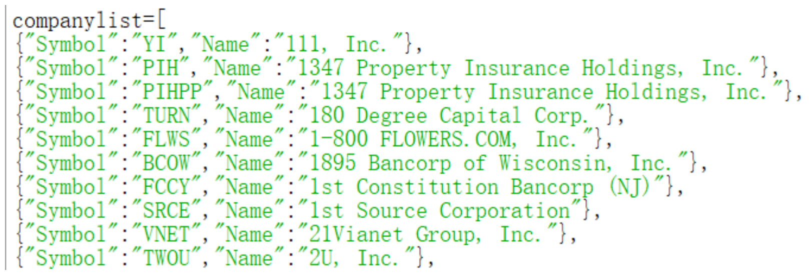
3. Have multi-round dialogue.

4. Deal with pending action.

The bot can also be deployed on Wechat and automatically respond to users' messages.

Details:

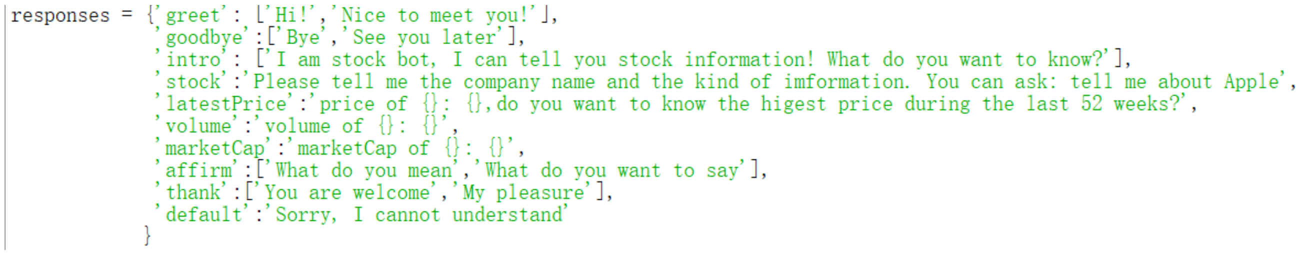
1. User log in: Check if the input id exists in the database "users.db".
2. Iexfinance is used to get stock data. Since Iexfinance requires a stock symbol to get stock data, a list of companies and their stock symbol is provided in the source code.



```
companylist=[
{"Symbol":"YI","Name":"111, Inc."},
{"Symbol":"PIH","Name":"1347 Property Insurance Holdings, Inc."},
{"Symbol":"PIHPP","Name":"1347 Property Insurance Holdings, Inc."},
{"Symbol":"TURN","Name":"180 Degree Capital Corp."},
{"Symbol":"FLWS","Name":"1-800 FLOWERS.COM, Inc."},
{"Symbol":"BCOW","Name":"1895 Bancorp of Wisconsin, Inc."},
{"Symbol":"FCCY","Name":"1st Constitution Bancorp (NJ)"},
{"Symbol":"SRCE","Name":"1st Source Corporation"},
{"Symbol":"VNET","Name":"21Vianet Group, Inc."},
{"Symbol":"TWOU","Name":"2U, Inc."},
```

The company list

3. Spacy is used to extract company names in user messages. Regular expression is used to extract date in user message and rasa_nlu is used to recognize intent. Since only a small amount of training data was provided, intent recognition with rasa_nlu was not accurate. Consequently, the module is only used for recognizing some kind of intent, and other kinds are recognized by detecting key words.
4. Responses are generated by adding data from iexfinance to defined formats.

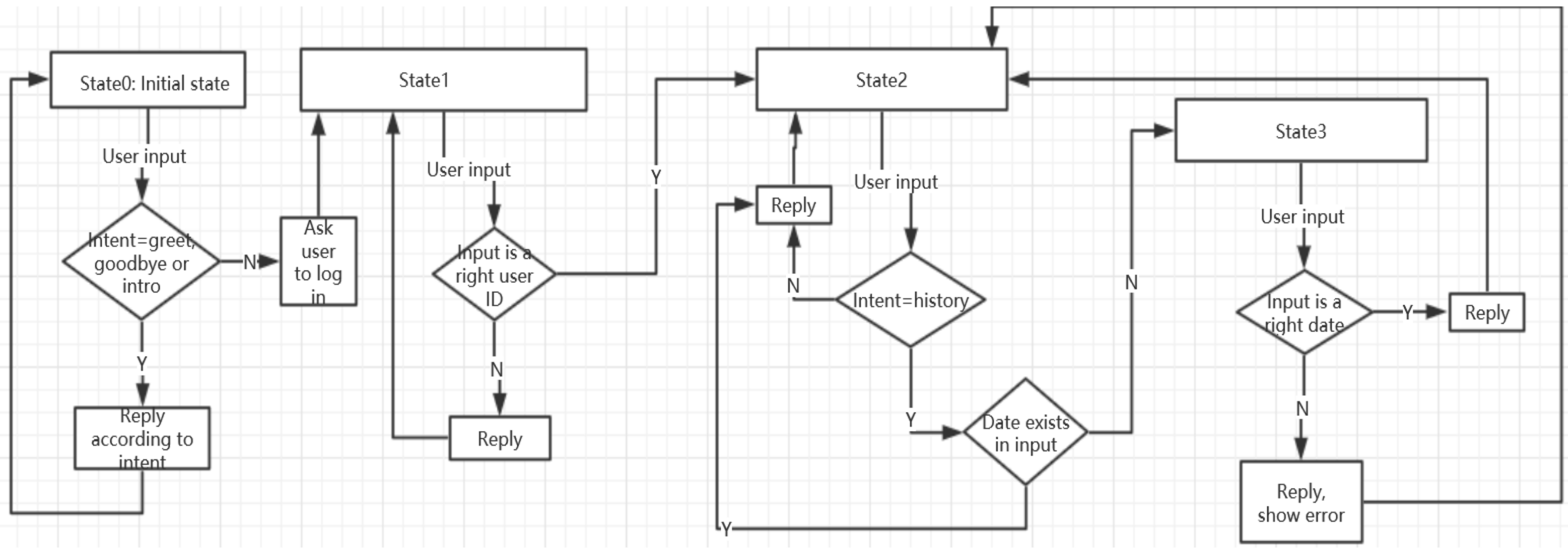


```
responses = {'greet': ['Hi!','Nice to meet you!'],
             'goodbye':['Bye','See you later'],
             'intro': ['I am stock bot, I can tell you stock information! What do you want to know?'],
             'stock':'Please tell me the company name and the kind of imformation. You can ask: tell me about Apple',
             'latestPrice':'price of {}: {},do you want to know the higest price during the last 52 weeks?',
             'volume':'volume of {}: {}',
             'marketCap':'marketCap of {}: {}',
             'affirm':['What do you mean','What do you want to say'],
             'thank':['You are welcome','My pleasure'],
             'default':'Sorry, I cannot understand'
            }
```

Format of responses

5. Wxpy module is used to deploy the bot on Wechat.

6. The state diagram for the state machine:



State diagram

4. Conclusion

During the research project, I learned to use spacy and rasa_nlu for natural language understanding. I also learned about some methods, including incremental slot filling and state machine, to build chat bots.

Though the research project did not involve training a machine learning model from the beginning and adjusting all parameters by myself, due to the need for huge amounts of data and computing, it let me learn a lot about machine learning. When using spacy, I found that with pre-trained word vectors, the results of finding word similarity, extracting entities and analyzing dependency can be reliable. However, when training a support vector machine with a training set of 1208 samples and testing it with a test set of 201 samples, I found it has an accuracy of about eighty percent-not very high. During the debug process of the final project, I found that since only a small amount of training data was provided, intent recognition with rasa_nlu was not accurate. Detecting key words is used for recognizing some kinds of intent instead. I realize that large amounts of data is important for machine learning.