Building a natural language chatbot based on Rasa NLU

Ke Xu

Abstract

In this project, we build a chabot whose framework are built on Rasa NLU, combining the spaCy and sklearn to classify intent and extract entities. In order to achieve single round query and multi round query, we also take the slot filling and finite state machine as the basic method of chatbot. What's more, we integrate the iexfinance API into this framework to get real-time stock information and achieve natural language conversation between user and chatbot by calling wxpy API.

***Keywords*: Rasa NLU, spaCy, machine learning, slot filling, finite state machine**

1 Introduction

A chatbot (also known as a smartbot, talkbot, Bot, IM bot, interactive agent, conversational interface or artificial conversational entity) is a computer program or an artificial intelligence which conducts a conversation via auditory or textual methods.[1] Such programs are often designed to convincingly simulate how a human would behave as a conversational partner, thereby passing the Turing test. Today, most business chatbots such as Google Assistant or Amazon Alexa are accessed via virtual assistants or via social software apps such as Facebook Messenger or WeChat.[2]

Natural language intent classification and named entities extraction has been the hotspot during the past decades due to its business potential. Methods including Regular Regression, Support Vector Machine, K-Nearest Neighbors, Neutral Network and Conditional Random Field has also been used to improve the efficiency of intent recognition or named entities extraction. The occurrence of word vectors has also prompt the effect of these methods. Natural language processing toolkits such as spaCy or Rasa NLU has integrated different kinds of machine learning methods to train the high quality model of intent classification and named entities extraction. With the help of these packages and combining methods such as slot filling or state machine, it is possible to build a chatbot with practical value.

2 Principle of Intent Classification

2.1 Training word vector by machine learning[3]

The occurrence of word vector is derived from the model of Neural Probabilistic Language Model (NLP) and Neural Network Language Model. Based on the model of Neural Probabilistic Language Model, semantic-based vector representation can improve the efficiency of NLP by analyzing the part of speech and dependency relationship of different word vectors. The basic principle of comparing the similarity of two word vectors is to compare the cosine angle of these two vectors. In order to train the Neural Network Language Model, we need to change the words in the vocabulary into the distributed feature vector of m-dimension, then represent probability functions with the distributed feature vector and train the model. As we can see, the occurrence of word vectors help us to achieve the goal of training natural language with neutral network.

2.1.1 Skip-Grams & Continuous Bag of Words of Word2vec[4]

Word2vec can be used as the input of Neural Probabilistic Language Model, in another word, it is the co-product and intermediate result of NPL. In order to predict the context and objective

word, we can separately use the model of Skip-grams and Continuous Bag of Words (CBOW). The main difference of these two models is that the Skip-Grams which fits for the large corpus predicts the sentences from the objective words, whereas, the CBOW which fits for the small corpus predicts the objective words from the sentences.

The basic principle of simple Skip-Grams can be seen in the figure:

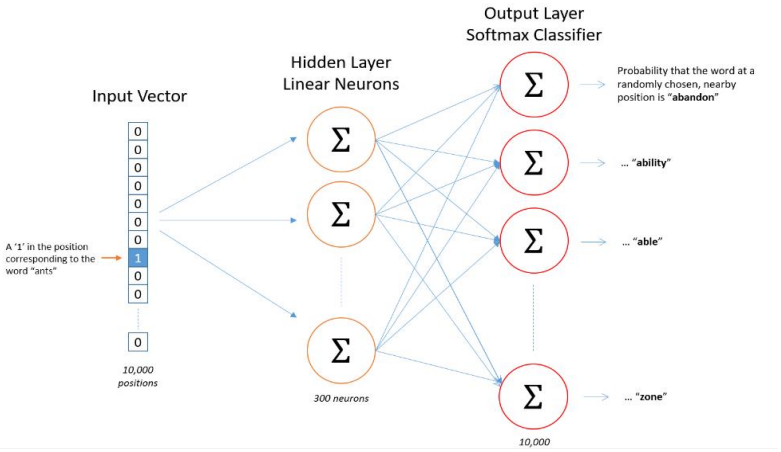

Fig.1 basic principle of simple Skip-Grams

The goal of Skip-Grams is to predict neighbor words' probability of objective words in the corpus. Input of this model needs to be changed into the word-pair and to be represented in the form of one-hot vector. First, we need to project the one-hot vector into the hidden layer and then multiply the results of hidden layer by the weight matrix of hidden layer to output layer, then normalize the results of output by softmax function:

$$y = \frac{e^{y}}{\sum i \epsilon y e^{i}} \tag{1}$$

y is a vector of $1\times |V|$, $|V|$ is the length of corpus, each components of vector represent the neighbor words' probability of objective word ω_i in the corpus. By multiplying the y by the output one-hot vector ω_o, we can get the output words' probability under the condition of objective word ω_i.

$$P(\omega_o|\omega_i) = y \times \omega_o \tag{2}$$

In order to optimize the data structure, we often replace neutron of the hidden layer to output with the Hierarchical Softmax.

Compared to the Skip-Grams, CBOW is a language model which can predict objective word by analyzing the context. Its goal is to maximize the logarithmic likelihood function:

$$\sum_{\omega_i \epsilon C} logp(\omega_i|context(\omega_i)) \tag{3}$$

The word vectors of context can be used as the input of CBOW, then add these word vectors together in the hidden layer and output the ω_i with the maximum logarithmic likelihood function. We often use Huffman Tree and Negative Samples to train this model.

2.1.2 Global Vectors (GloVe) [5]

The reason why this model is called Global Vectors is that the global corpus statistics are captured directly by the model. First, we need to establish the matrix of word-word co-occurrence denoted by X_{ij} whose entries X_{ij} tabulate the number of times word *j* occurs in the context of word *i*. Second, we use the weighted least squares regression model to train the word vectors.

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(\omega_i^T \overline{\omega_j} + b_i + \overline{b_j} - logX_{ij}\right)^2 \tag{4}$$

$f(X_{ij})$ is the weighting function which can be parameterized as:

$$f(x) = f(x) = \begin{cases} (x/x_{max})^{\alpha} & x < x_{max} \\ 1 & otherwise \end{cases} \quad (5)$$

ω_i and ω_j are the word vectors of word *i* and *j*; b_i and b_j are additional bias.

Model of CBOW or Skip-Grams is a local context window which lack of the relationship between global words to local words, and miss the relationship information by Negative Samples. Unlike these two models, GloVe is a model that utilizes benefit of count data while simultaneously capturing the meaningful linear substructures prevalent in recent log-bilinear prediction-based methods like word2vec. As a result, GloVe, is a global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks.

2.1.3 FastText[6]

FastText is a compression text classification models introduced by Facebook AI research. This method outperforms a good margin in terms of the compromise between memory usage and accuracy compared to the SVM, NN method. The framework of FastText is similar to the CBOW, which is also based on the Hierarchical Softmax. The main difference between CBOW and FastText is in following two aspects:

Output of model: the output of word2vector is the probability of each term, whereas the output of FastText is the label of classification.

Input of model: the input of word2vector is the term of windows context, but the input of FastText is the whole sentence including the term.

2.2 Intent Classification by supervised learning: Support Vector Machine[7]

Usually, corpus need to be transferred into the form of word vectors before they are trained by the SVM. Marcelo Mendoza proposed three kinds of vector representation separately based on descriptive text and clicks, descriptive text and reading time and descriptive text, reading time and clicks. Due to the large term collections and necessity of classifying high-dimensional data effectively, SVM was introduced to their work. In this context, the support vector machine (SVM) have proven to be useful in processing high-dimensional vectors (over 10^6 features) even when the vectors are sparse, such as in the case of text. The machine learning algorithm that Rasa NLU uses to classify intents of sentences is the SVM with GridSearchCV.

The SVM in Scikit-learn packages of Python can also be used to classify the intent combining with spaCy. Before training the SVM model, corpus should be transferred into word vectors by spaCy and word vectors should also be changed into the data form of *.csv in order to be read by SVM. The code can be shown as:

```
# 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)
```

2.3 Intent Classification by unsupervised learning: K-Nearest Neighbors[8]

Various classification methods have been used for intent classification. K-Nearest Neighbors can operate with minimal computational effort and training data, which makes it appropriate for real time implementation.

K-NN usually includes two steps: (1) Given a word vector, find its nearest neighbors in the training data based on some similarity metric; (2) Identify the class of the word vector by determining the maximum number of classes among the nearest neighbors. Normalized Euclidean distance is the typical similarity metric.

3 Principle of Named Entity Recognition[9]

NER can be defined as the task that attempts to locate, extract, and automatically classify named entities into predefined classes or types in open-domain and unstructured texts, such as newspaper articles. It is essential to recognize information units like names, person, organization and location, numeric expressions like time, date, money. One obvious difficulty of named entities is their pervasiveness, which is defined as the high frequency such as occurrence and co-occurrence of named entities in corpora.

3.1 Supervised learning

The current dominant technique for solving the Named Entity Recognition problem is supervised learning. Supervised learning techniques include Decision Trees (S. Sekine 1998), Support Vector Machines (SVM) (M. Asahara & Matsumoto 2003), and Conditional Random Fields (CRF) (A. McCallum & Li 2003). All these supervised learning approach includes a system that reads a large amount of annotated corpus, memorizes lists of entities, and creates disambiguation rules based on discriminative features.

A baseline supervised learning method consists of tagging words of a test corpus when they are annotated as entities in the training corpus. The performance of the baseline system depends on the vocabulary transfer, which is the proportion of words, without repetition, appearing in both training and testing corpus.

3.2 Semi-supervised learning

The term “semi-supervised” is present recently. The main technique for semi-supervised learning is called “bootstrapping”. It involves a small degree of supervision. For example, a system aimed at finding “names” needs a small number of example names before training. Then the system searches for sentences that contain these names and tries to recognize some textual clues common to the examples. After that, the system tries to find other examples of names that appear in similar contexts. The process is then reapplied to the newly found examples in order to find new relevant contexts. By repeating this process, a large number of names and a large number of contexts will eventually be gathered.

3.3 Unsupervised learning

The typical approach in unsupervised learning is clustering. For example, one can gather named entities from clustered groups based on the similarity of context. This technique rely on textual resources, textual patterns and is determined on a large unannotated corpus.

4 Intent Classification and Named Entity Recognition in Rasa NLU

There are two main pipeline in the Rasa NLU. “spacy_sklearn” is a spaCy model which is appropriate to train the corpus less than 1000 examples, whereas, “tensorflow_embedding” is

suitable to train the corpus more than 1000 examples. The biggest difference between these two method is that “spacy_sklearn” uses pre-trained word vectors from either GloVe or fast Text, the “tensorflow embedding” does not use any pre-trained word vectors, but fits for your own defined dataset. “spacy_sklearn” is useful if you do not have many training data. The advantage of “tensorflow_embedding” pipeline is that your word vectors will be customised for your domain. This pipeline doesn’t use a language-specific model, so it will work with any language that you can tokenize.

Rasa NLU supports multiple intent classifiers like sklearn, mitie and tensorflow. we adopt sklearn with classical SVM classifier, since deep learning methods need a larger training set to give satisfactory accuracy. We use spaCy to tokenize text to words and convert words to feature vectors. Then, we label these feature vectors and train the SVM with GridSearchCV to classify intents.

These four method are used in Rasa NLU to extract entities, and we can also use regular expression to help CRF model extract entities. To match regular expression exactly, we can do this process in our code after receiving responses from Rasa NLU.

Component	Requires	Model	notes
ner_crf	sklearn-crfsuite	conditional random field	training custom entities
ner_spacy	spaCy	averaged perceptron	pre-trained entities
ner_duckling_http	running duckling	context-free grammar	pre-trained entities
ner-mitie	MITIE	structured SVM	training custom entities

Table. 1 entity extraction method in Rasa NLU

5 Implementation of Chatbot

In this part, we will discuss the principles of Slot Filling and State Machine and how to build the Chatbot with them.

5.1 Incremental Slot Filling[10]

Incremental slot filling can be used to design a conversation flow for parameters value collection within a single intent every time. This method is useful under the situation that an action can't be completed without a specific set of parameter values. For example, we set some parameter values to be required by the users. If users omit one or more parameters in their response, our chatbot will require them to provide more values of the missing parameter. To use incremental slot filling, we need to create pre-defined entities database in the form of json or sqlite. Then when user send message into the chatbot, we recognize the intents and extract the entities of the message. Then judge the missing parameters and require user to provide more specific information. Incremental slot filling can be used to design FAQ (Frequently Asked Questions)-Chatbot.

In this project, we use sqlite to store the pre-defined parameters and create the query of parameters and negation parameters:

```
# Define find_stocks()
# find the parameters and negation parameters
def find_stocks(params, neg_params):
    # create the query
    query = 'SELECT * FROM stocks'
```

```
    # create the query if parameters and negation parameters exists
    if len(params) > 0 and len(neg_params) > 0:
        filters = ["{}=?".format(k) for k in params] + ["{}!=?".format(k) for k in neg_params]
        query += " WHERE " + " and ".join(filters)
    # create the query if negation parameters exists
    elif len(neg_params) > 0:
        filters = ["{}!=?".format(k) for k in neg_params]
        query += " WHERE " + " and ".join(filters)
    # create the query if parameters exists
    elif len(params) > 0:
        filters = ["{}=?".format(k) for k in params]
        query += " WHERE " + " and ".join(filters)
    # define the parameters and negation parameters
    t = tuple(dict(list(params.items()) + list(neg_params.items())).values())
    # open connection to .db
    conn = sqlite3.connect('stocks.db')
    # create a cursor
    c = conn.cursor()
    # execute the query
    c.execute(query, t)
    # get the result
    return c.fetchall()
```

Find negation parameters:

```
# Define find negation_parameters()
def find negation_parameters (message, parameters):
    # extract all parameters in the message
    params = [p for p in parameters if p in message]
    # cut the message and find end of each clause
    ends = sorted ([message.index(p) + len(p) for p in params])
    start = 0
    # create chunks[] to hold all clausees
    chunks = []
    for end in ends:
        chunks.append (message[start:end])
        start = end
    # create results{} to hold the judgment results
    result = {}
    # to judge whether negation parameters in the chunks
    for chunk in chunks:
        for param in params:
            if param in chunk:
                if "not" in chunk or "n't" in chunk:
```

```
                result [param] = False
            else:
                result [param] = True
    return result
```

Then we use Rasa NLU to recognize the intent and extract entities of user's questions and create respond functions.

```
# Define the respond_find_stocks function
def respond_find_stocks (message):
    # Extract the entities
    entities = interpreter.parse(message)["entities"]
    ent_vals = [e["value"] for e in entities]
    # Look for negated entities
    negated = negation_parameters (message, ent_vals)
    for ent in entities:
        if ent["value"] in negated and not negated[ent["value"]]:
            neg_params[ent["entity"]] = str(ent["value"])
        else:
            params[ent["entity"]] = str(ent["value"])
    # Find the stocks
    results = find_stocks(params, neg_params)
    names = [r[0] for r in results]
    n = min(len(results),3)
    # Select the nth element of the responses array
    return responses[n].format(*names)

# Initialize params and neg_params
params = {}
neg_params = {}

# Define responses
responses = ['I'm sorry :( I couldn't find anything like that',
             '{} is a stock you are looking for!',
             '{} or {} would answer your question!',
             '{} is one option, I can tell you more :)']
```

5.2 Finite State Machine

Finite State Machine is a model of computation based on a hypothetical machine made of one or more states. Only a single state can be active and every state represents an action at the same time, so the machine must transit from one state to another state to perform different actions.

In this project, we use following policy rules to achieve state machine:

```
# Define the state
STATE0 = 0
STATE1 = 1
STATE2 = 2

# Define the policy rules
policy_rules = {
                    (0, "intent1"): (0, respond_sentence("intent1"), None),
                    (0, "intent2"): (0, respond_sentence("intent2")[0], 1),
                    (0, "intent3"): (1, respond_sentence("intent3"), None),
                    (0, "intent2"): (2, respond_sentence("intent2")[1], None)
                    }
```

Then use the following function to achieve the state transition and to deal with pending action:

```
# Define send_message()
def send_message(state, pending, message):
    # calculate the new_state, response, and pending_state
    new_state, response, pending_state = respond(state, message)
    # judge the pending
    if pending is not None:
        # do pending: calculate the new_state, response, and pending_state
        new_state, response, pending_state = policy_rules[pending]
        # clear the pending
        if response is not None:
            return new_state, None
    # create the new pending
    if pending_state is not None:
        pending = (pending_state, match_intent(message))
    return new_state, pending

# Define respond()
def respond(state, message):
    entity = match_entity(message)
    # calculate the new_state
    new_state = policy_rules[(state, match_intent(message))][0]
    # create the pending_state
    pending_state = policy_rules[(state, match_intent(message))][2]
    # response of intent1
    if match_intent(message) == 'intent1':
        response = policy_rules[(state, match_intent(message))][1]
    return new_state, response, pending_state
```

Both match_entity(message) function and match_intent(message) function are achieved by the Rasa NLU.

6 Application of Chatbot

We use the established chatbot to achieve real time stock information query and deploy it into social software WeChat.

6.1 iexfinance API

Iexfinance is an easy-to-use package which can obtain stock information including: real-time and delayed quotes, historical data, financial statements, sector performance, market analysis, etc. What's more, it can also plot all kinds of stock data with matplotlib.

The examples can be seen as:

```
# Import the Stock
from iexfinance. stocks import Stock
# Get real-time price
tsla = Stock('TSLA')
tsla. get_price()
```

6.2 wxpy API

Wxpy is an API of WeChat that is the most popular social software in China. We connect the chatbot with WeChat by the wxpy in order to achieve the auto reply function. Package wxpy needs to be installed and imported into the project first. A QR code will be generated after creating a bot. Then we need to define an auto reply function to achieve interaction between human and chatbot.

The function can be achieved in the following:

```
# Import the wxpy
from wxpy import *
# Initialize the bot
bot = Bot()
# Find the friend
my_friend = bot.friends(). search('name')
# Send the message
my_friend.send()
# Send the image
my_friend.send_image()
# Define auto reply function
@bot.register(my_friend)
def auto_reply(msg):
    return 'received: {}'.format(msg.text)
```

Specific installation and configuration information can be seen on the README.md of https://github.com/Tknight01/Ke

7 Conclusion

Based on the discussion above, we introduce a chatbot based on Rasa NLU and deploy this chatbot into WeChat.

(1) “spacy_sklearn” pipeline can be used to classify intents and extract entities of stock information query effectively by pre-defining corpus training data.

(2) slot filling and finite state machine can be integrated into the built Rasa NLU framework to achieve natural language conversation; the extraction of negation entities and transfer of pending action can also be proved effectively to deal with the deny intent and message queue.

(3) integrating iexfinance and wxpy into our chatbot can functionalize our chatbot and make our chatbot more practical.

Acknowledgement

The author is grateful to Ph.D. Fan Zhang for the fruitful comments and instruction on this subject.

Reference

[1] Margaret Rouse. *"What is a chatbot?"*, Retrieved from techtarget.com.

[2] Orf, Darren. *"Google Assistant Is a Mega AI Bot That Wants To Be Absoutely Everywhere"*. Retrieved from gizmodo.com.

[3] AM Rahman, Abdullah Al Mamun, et al. *“Programming challenges of Chatbot: Current and Future Prospective”*, 2017 IEEE Region 10 Humanitarian Technology Conference, 2017

[4] David Guthrie, Ben Allison, et al. *“A Closer Look at Skip-gram Modelling”*, 2006.

[5] Jeffrey Pennington and Richard Socher, et al. *“Glove: Global vectors for word representation.”* Proceedings of the Empiricial Methods in Natural Language Processing, 2014.

[6] Joulin, Armand, et al. *"FastText.zip: Compressing text classification models."* , 2016.

[7] Mendoza, Marcelo, and J. Zamora. *"Identifying the Intent of a User Query Using Support Vector Machines."* Lecture Notes in Computer Science, 2009.

[8] Khademi, Gholamreza, et al. *"Evolutionary optimization of user intent recognition for transfemoral amputees."* Biomedical Circuits & Systems Conference, 2015.

[9] Nadeau, David, and S. Sekine. *"A survey of named entity recognition and classification."* Lingvisticae Investigationes, 2007.

[10] Xu, Puyang, and R. Sarikaya. *"Convolutional neural network based triangular CRF for joint intent detection and slot filling."* IEEE Workshop on Automatic Speech Recognition & Understanding, 2014.