Description

(a) Brief introduction of our method

Our dialogue system is tasked-based. It contains mainly two modules incl. understanding and classifying input questions, as well as managing dialogue and generating corresponding answers. The understanding and classification of input questions are based on TF-IDF (term frequency-inverse document frequency) algorithm. We have also invented some data as back-end database to our system. The system can realize a continuous dialogue session and help with some classes of questions presented in the database. In the following paragraph, we will briefly discuss each module in our system.

Back-end data

We have manipulated two tables to support our system. One is 'questions' table and the other one is 'answers' table. In 'questions' table, it contains several questions together with their corresponding class. For example, 'how is the weather', 'what is the temperature', 'what is the weather like' all represent the question class of 'weather', same as other classes of 'restaurant' and 'bus'. We've also created two classes as 'ending' and 'thank' representing the greeting terms in normal conversation. In 'answers' table, we have created several answers to the different classes with respect the corresponding cities. For example, in city 'Gothenburg', the weather is 'sunny, 10 degrees celsius', the restaurant is 'SWEDISH TASTE' and the bus is '20 min later at A bus station'.

Understand and classify the input questions

Here we are using the TF-IDF (term frequency-inverse document frequency) algorithm to measure the similarity between the input question and each of the questions in the database. We will classify the input question same as the one in the database most similar with.

We firstly calculate TF which represents the word frequency. If both sentences contain similar words, then we assume they are similar. We split the input sentence and compared sentence into words, abstract the set union of these words and put them in a 'words bag'. We use dictionary key-value pair to record the word sequence. Then we count the words in each sentence and vectorize the words frequency. For example, we have the input sentence as 'how is the weather' and the compared sentence as 'what is the weather like'. The generated words bag is [how, is, the, weather, what, like]. The words frequency vectors for each of sentences are [1,1,1,1,0,0] and [0,1,1,1,1,1] respectively.

As we have discussed, if two sentences contain similar words, then they are similar. However, this is not always the case. For example, the common words or so-called stop words as ‘the’, ‘and’, ‘is’ can appear quite often in many sentences, while they cannot represent the actual similarities of two sentences. Instead, if a word is not common in the whole database while appearing in the input sentence, then this word can probably represent the sentence’s content. IDF (inverse document frequency) is introduced to solve this problem. It adds extra weight to each word by the following formula,

$$IDF = log(\frac{number\ of\ sentences\ in\ the\ database}{number\ of\ sentences\ containing\ the\ word + 1})$$

From the formula, we can see that if a word appears more often in the database, then the denominator is bigger and its IDF value is smaller. On the other hand, if a word appears less often in the database, then the denominator is smaller and its IDF value is bigger. In this way, the IDF term adds smaller weights to common words and bigger weights to uncommon words while representing the sentence’s content. Then, we revise both words’ frequency vectors obtained by TF with IDF through element-wise multiplication.

Finally, we calculate the similarity between two words’ frequency vectors using cosine similarity equation. Cosine similarity is measured by the cosine value of angle between the two vectors, the bigger the cosine value, the more similar the two vectors are. After calculating the similarity values of input sentence between all sentences in the database we pick the most similar sentence and assign its class label to the input question. Here we also set a threshold of 0.5 to indicate no similar sentence in the database. The threshold is chosen through several experiments, and 0.5 performs well to filter those non-similar sentences.

Manage dialogue and generate corresponding answers

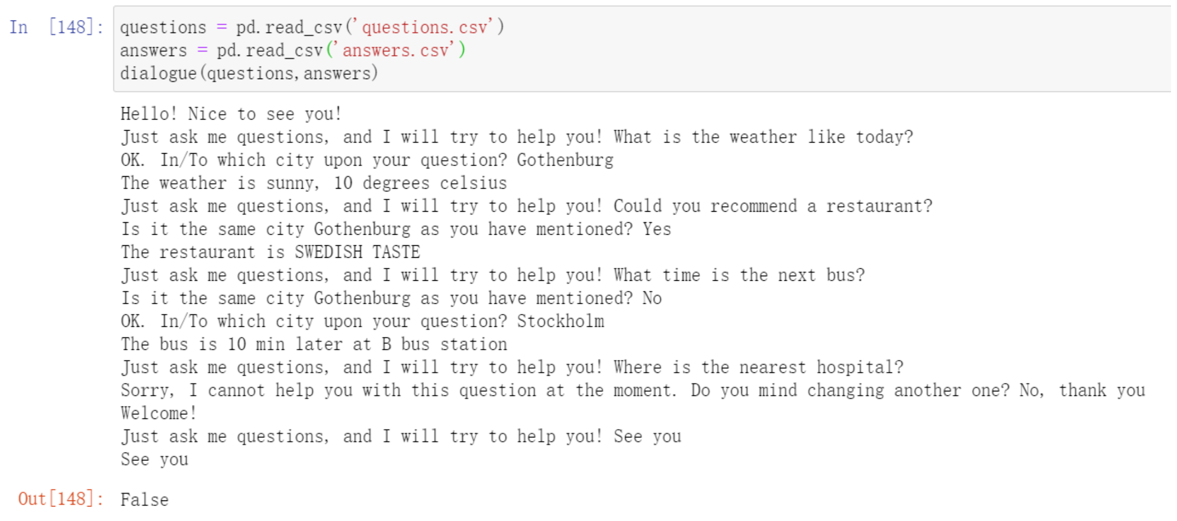
The system can track the question class from the previous user input, asking for user’s further input of city, and the system will output the answer matching with the class of question from the ‘answers’ database with respect to the input city. We also deal with several special cases and exceptions when answering the questions.

- When the input sentence cannot match with any of the questions in the ‘questions’ database, the system will ask the user to input another question.
- Once the user has input a city, the system will remember the city and output answers with respect to that city if the user wants the information for the same city in the next question. Otherwise, the system will ask the user to input another city.
- Once the user expresses gratefulness such as ‘thank you’, the system will feedback with ‘Welcome’ as politeness.
- Once the user wants to finish the conversion by saying such as ‘see you’, the system will reply with the same words and end the dialogue session.

Final illustration of a dialogue session

```
Machine: Hello! Nice to see you!
Just ask me questions, and I will try to help you!
```

Me: What is the weather like today?
Machine: OK. In/To which city upon your question?
Me: Gothenburg
Machine: The weather is sunny, 10 degrees celsius
Just ask me questions, and I will try to help you!
Me: Could you recommend a restaurant?
Machine: Is it the same city Gothenburg as you have mentioned?
Me: Yes
Machine: The restaurant is SWEDISH TASTE
Just ask me questions, and I will try to help you!
Me: What time is the next bus?
Machine: Is it the same city Gothenburg as you have mentioned?
Me: No
Machine: OK. In/To which city upon your question?
Me: Stockholm
Machine: The bus is 10 min later at B bus station
Just ask me questions, and I will try to help you!
Machine: Where is the nearest hospital?
Sorry, I cannot help you with this question at the moment. Do you mind changing another one?
Me: No, thank you
Machine: Welcome!
Just ask me questions, and I will try to help you!
Me: See you
Machine: See you



```
In [148]: questions = pd.read_csv('questions.csv')
          answers = pd.read_csv('answers.csv')
          dialogue(questions,answers)
```

```
Hello! Nice to see you!
Just ask me questions, and I will try to help you! What is the weather like today?
OK. In/To which city upon your question? Gothenburg
The weather is sunny, 10 degrees celsius
Just ask me questions, and I will try to help you! Could you recommend a restaurant?
Is it the same city Gothenburg as you have mentioned? Yes
The restaurant is SWEDISH TASTE
Just ask me questions, and I will try to help you! What time is the next bus?
Is it the same city Gothenburg as you have mentioned? No
OK. In/To which city upon your question? Stockholm
The bus is 10 min later at B bus station
Just ask me questions, and I will try to help you! Where is the nearest hospital?
Sorry, I cannot help you with this question at the moment. Do you mind changing another one? No, thank you
Welcome!
Just ask me questions, and I will try to help you! See you
See you

Out[148]: False
```

Several limitations of our system

- The questions classification is totally based on the questions database. If the database is small, it's hard to classify the question into an appropriate class. Besides, our method to measure the sentence similarities depends much on the words' frequency. This can be less flexible as there are normally many kinds of ways to express the same meaning through language. Of course, this can also be improved by a bigger database.

- Even with a bigger database, it may be still hard to track those hidden/indirect expressions in human language. For example, if the user inputs sentence like 'I am sick and want some treatment' instead of 'where is the hospital', then it's difficult for the system to capture the real meaning of the user.
- Our system has a limited memory to the history conversation. It cannot track the long-term dialogue.
- The language generation of our system is not flexible. It only outputs the answers searched from the answers database without anything else.
- Our system is rule-based and not so efficient, the more complex our system is, the more manual programming work is needed.

(b) Discussions on further improvements

- In the user input understanding and classification module, we could search for more dialogue material as the training data, and train machine learning model which can predict the domain and intent of the use input.
- We could use a statistical dialogue management module with dialog state tracking and policy learning. This is based on partially observed Markov Decision Process, maintaining a distribution over multiple hypotheses of the true dialog state. The optimal action from the current state maximizes the expected reward.
- In the answer generating module, we could try with the recurrent neural network (RNN) model to produce words in sequence to form the machine response. This can make the response more flexible and closer to human language.