Development doc for chatbot

Project overview

```
.
├── chatbot_config.py           # configuration file for our chatbot
├── database                    # mysql configuration files
│   ├── docker-compose.yaml     # used to run a mysql container on my computer for devel
│   └── mysql_data              # equals to the files in /var/lib/mysql
├── document                    # document of this project
├── lib                         # chatbot library
│   ├── abstract_status.py      # an abstract class (or Interface in Java) used for obje
│   ├── intent_classificion.py  # functions used for users' intent classification
│   ├── recommendation.py       # functions used for recommending specific content based
│   └── status.py               # a child class of the AbstractStatus
├── main.py                     # chatbot entrance
├── meta                        # some useful objects saved using Pickle
│   ├── destination_df.pk
│   ├── intent_classification_svc.pk
│   ├── review_df.pk
│   └── toplist_df.pk
├── spider                      # root directory for scrapy
│   ├── trip_com                # scripts used for grabing content from trip.com
│   │   ├── scrapy.cfg
│   │   └── trip_com
├── templates                   # html pages, css, javascript fils used in the frontend
│   ├── Stats.js
│   ├── TweenMax.min.js
│   ├── chatbox.html
│   ├── css
│   ├── css_globe_PerspectiveTransform.js
│   ├── dat.gui.min.js
│   ├── fonts
│   ├── images
│   ├── index.html
│   ├── index.html.bak
│   ├── js
│   ├── modernizr.min.js
│   ├── script.js
│   ├── style.css
│   └── test.html
└── utils.py                    # some useful functions
```

Step 1 of our project --- grabbing data

> library: `scrapy`
> link: `https://scrapy.org/`
> code examples:

```
# this function was used to grab the content from
# https://www.trip.com/travel-guide/inspiration-list/
def inspiration_parser(self, response, **kwargs):

    toplists = response.css('[class="jsx-3344324880 toplist-content"]')
    toplist_items = list()
    for cur_toplist in toplists:
        name = cur_toplist.css('[class="jsx-3344324880 title"]')\
            .css('a.jsx-3344324880::text').get()
        link = cur_toplist.css('[class="jsx-3344324880 more-icon"]')\
            .css("a.jsx-3344324880::attr(href)").get()
        toplist_info = {"name": name, "link": link}
        toplist_items.append(toplist_info)
        yield scrapy.Request(link, self.toplist_parser, meta={"inspiration": name})

    yield {"inspirations": toplist_items, "type": "inspirations"}


# this function was used to grab the content from webpages like
# https://www.trip.com/travel-guide/toplist-569/
def toplist_parser(self, response, **kwargs):

    res = dict()
    res["type"] = "toplists"
    res["inspiration"] = response.meta["inspiration"]
    res["destinations"] = list()
    destination_list = response.css('[class="jsx-946890623 item"]')

    for cur_destination in destination_list:

        cur_destination_info = dict()

        cur_destination_name = cur_destination.css('[class="jsx-946890623 content-title'
            .css("a.jsx-946890623::attr(title)").get()

        cur_destination_link = cur_destination.css('[class="jsx-946890623 content-title'
            .css("a.jsx-946890623::attr(href)").get()

        cur_destination_description = cur_destination.css('[class="jsx-946890623 conten

        cur_destination_info["name"] = cur_destination_name
        cur_destination_info["link"] = cur_destination_link
        cur_destination_info["description"] = cur_destination_description

        res["destinations"].append(cur_destination_info)

        yield scrapy.Request(cur_destination_info["link"], callback=self.destination_pa
                             meta={"toplist": res["inspiration"]})

    yield res
```

> Collected content looks like this:

```
{
  "inspirations": {
    "inspirations": [
      {
        "name": "Best Asia-Pacific Travel Destinations",
        "link": "https://www.trip.com/travel-guide/toplist-569/"
      },
      {
        "name": "Celebrate Chinese New Year in Asia",
        "link": "https://www.trip.com/travel-guide/toplist-162/"
      },
      {
        "name": "The World's Most Beautiful Self-Driving Tours",
        "link": "https://www.trip.com/travel-guide/toplist-580/"
      }
    ],
    "type": "inspirations"
  },
  "toplists": {
    "type": "toplists",
    "inspiration": "Best Asia-Pacific Travel Destinations",
    "destinations": [
      {
        "name": "Margaret River, Australia",
        "link": "https://www.trip.com/travel-guide/margaret-river-1606/",
        "description": [
          "Don't miss the beautiful sea cliffs at Meelup Beach",
          " ",
          " ",
          "Top wineries, craft breweries, and the best food"
        ]
      },
      {
        "name": "Takamatsu, Japan",
        "link": "https://www.trip.com/travel-guide/takamatsu-56911/",
        "description": [
          "Tour the 88 beautiful temples on Shikoku Island",
          " ",
          " ",
          "Attend the Setouchi International Arts Festival"
        ]
      }
    ]
  },
  "destination": {
    "type": "destination",
    "toplist": "Best Asia-Pacific Travel Destinations",
    "name": "Nadi",
    "loc_tags": "Trip, Travel Guides, Oceania, Fiji, Western Division, Nadi",
    "navigation": {
      "Attractions": "https://www.trip.com/travel-guide/nadi-1339/tourist-attractions/"
```

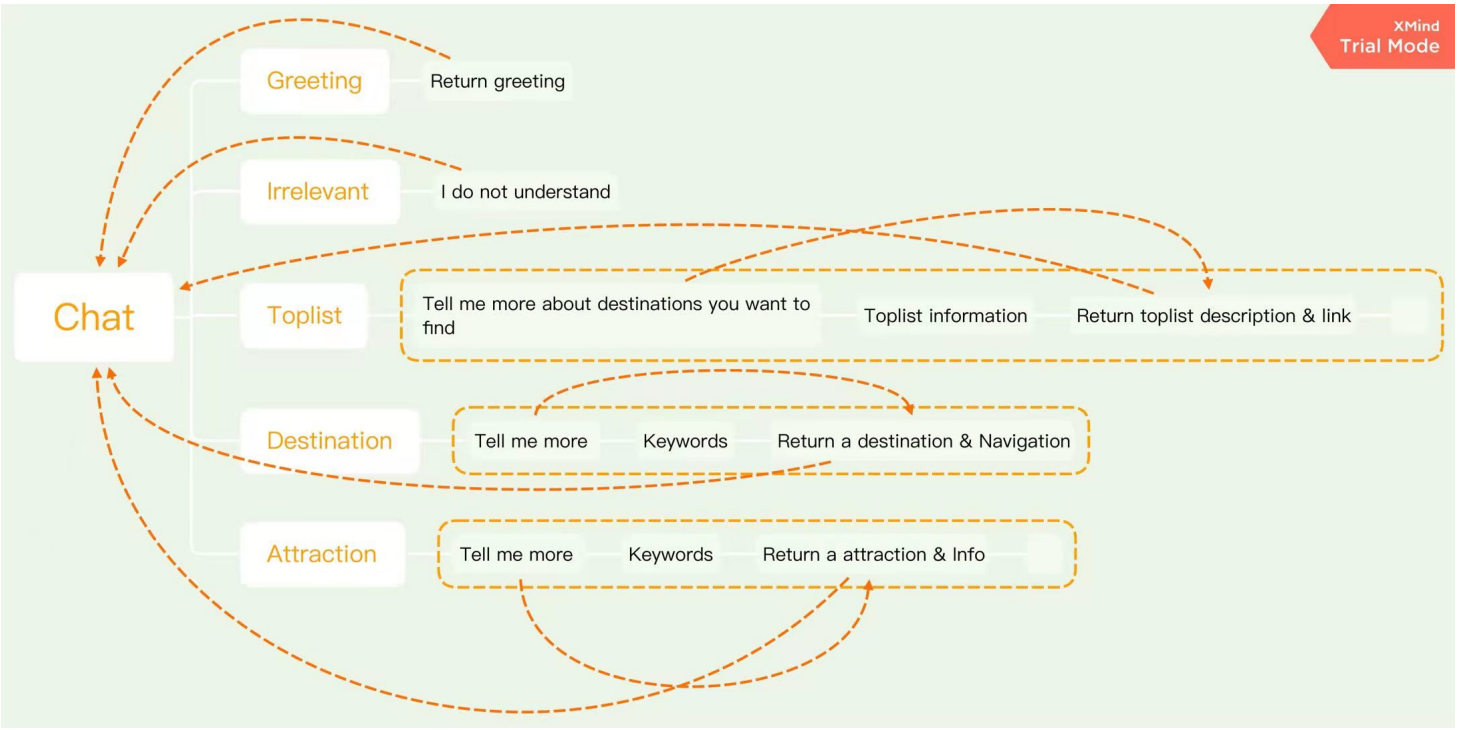
```
      "Hotels": "https://www.trip.com/hotels/list?city=791",
      "Restaurants": "https://www.trip.com/travel-guide/nadi-1339/restaurants/",
      "Tours/Tickets": "https://www.trip.comhttps://us.trip.com/things-to-do/list?searcl
      "Trip Moments": "https://www.trip.com/travel-guide/nadi-1339/image/"
    }
  },
  "attractions": {
    "type": "attractions",
    "attractions": [
      {
        "name": "The Palace Museum",
        "link": "https://www.trip.com/travel-guide/beijing/the-palace-museum-75595/"
      },
      {
        "name": "Badaling Great Wall",
        "link": "https://www.trip.com/travel-guide/beijing/badaling-great-wall-75596/"
      }
    ],
    "destination": "Beijing"
  },
  "attraction": {
    "type": "attraction",
    "destination": "Hiroshima",
    "loc_tags": "Trip, Travel Guides, Asia, Japan, Hiroshima, Hiroshima, Mount Misen",
    "name": "Mount Misen",
    "open_status": "Open 24 hours",
    "recommended_sightseeing_time": "2-3 hours",
    "phone": "+81-829-309141",
    "address": "Miyajimacho, Hatsukaichi, Hiroshima Prefecture",
    "reviews": [
      "Mount Misen is the sacred mountain on Itsukushima in Hatsukaichi, Hiroshima, Japa
      "Miyama is behind Itsukushima Shrine and is a world natural heritage. Walking fron
      "Mount Mishan is 535m above sea level and is the main peak of Miyajima. Because it
      "Miyama, Japan\u2019s three scenic spots, took a car to the Shishi Rock Observatio
      "This mountain is not very high, but standing on the top of the mountain, you can
    ]
  }
}
```


Then I stored the grabbed content in mysql. Tables are like this:

```
mysql> show tables;
+----------------------------+
| Tables_in_trip_info        |
+----------------------------+
| attraction                 |
| attraction_and_review      |
| destination                |
| destination_and_attraction |
| review                     |
| toplist                    |
| toplist_and_destination    |
+----------------------------+
7 rows in set (0.01 sec)
```

Step 2 of our project --- chatbot logic

The pipline of our chatbot can be represented by the following graph:



To implement the logic in the figure, we need to define a abstract class and several child classes to represent `Status`

Here is the code of `AbstractStatus`

```
class AbstractStatus(metaclass=ABCMeta):
    @abstractmethod
    def __init__(self):
        pass

    @abstractmethod
    def make_response(self):
        pass
```

Here is the code of the `StartStatus` or `InitialStatus`

```
class StartStatus(AbstractStatus):
    def __init__(self):
        self.type = "StartStatus"
        self.end_topic = False

    def make_response(self):
        return ""
```

Here is the transformation logic:

```
def next_status(cur_status, user_input):
    try:
        new_status = None
        if cur_status.type == "StartStatus":
            intent_idx, intent_name = intent_classificion.inference(intent_classificati
            if intent_name == "greeting":
                new_status = GreetingStatus()
                return new_status
            if intent_name == "toplist":
                new_status = ToplistStatus1()
                return new_status
            if intent_name == "destination":
                new_status = DestinationStatus1()
                return new_status
            if intent_name == "attraction":
                new_status = AttractionStatus1()
                return new_status
            if intent_name == "irrelevant":
                new_status = IrrelevantStatus()
                return new_status
        if cur_status.type == "ToplistStatus1":
            new_status = ToplistStatus2(toplist_df=toplist_df,
                                        sentence=user_input,
                                        w2v_model=w2v_model,
                                        db_connector=db_connector)
            return new_status
        if cur_status.type == "DestinationStatus1":
            new_status = DestinationStatus2(destination_df=destination_df,
                                            sentence=user_input,
                                            w2v_model=w2v_model,
                                            db_connector=db_connector)
            return new_status
        if cur_status.type == "AttractionStatus1":
            new_status = AttractionStatus2(review_df=review_df,
                                           sentence=user_input,
                                           w2v_model=w2v_model,
                                           db_connector=db_connector)
            return new_status
        return ErrorStatus() if new_status is None else new_status
    except:
        new_status = ErrorStatus()
        return new_status
```

And here is the transformation process.

```
@app.route("/get_response", methods=["POST"])
def get_bot_response():
    chat_history = utils.convert_request_form_to_list(request.form)

    global cur_status
    cur_status = next_status(cur_status, chat_history[-1]["content"])
    response = cur_status.make_response()

    if cur_status.end_topic:
        cur_status = StartStatus()

    return jsonify(response)
```

Step 3 of our project --- intent classification

When users input a text, we need to figure out what they want to do. So the first step of our chatbot is to train a classifier.
Based on our application area (Traveling), we can group users' intents into the following classes.

```
greeting      # She want to say hi to you
toplist       # She wants you to suggest her a list of places to go
destination   # She wants you to recommend her a city or a island to go
attraction    # She wants you to tell her an attraction to go
irrelevant    # She said something not in our scope
```

To train such a classifier, we need data. But this kind of data is hard to collect.
When data is insufficient, we have two ways to train a classifier:

> 1. Few shot learning
> 2. Data augmentation --- I used this method.

Data augmentation is commonly used in computer vision, especially image classification. We can use the insight from that. Here is a example text augmentation function.

```
def sentence_augmentation(sentence, w2v_model, topn=30, threshold=0.9, expected_count=20
    ret_list = list()
    for w in nltk.word_tokenize(sentence):
        if not w in w2v_model:
            continue
        similar_words = [cur_pair[0] for cur_pair in w2v_model.similar_by_word(w) if cur
        if similar_words:
            ret_list.extend([sentence.replace(w, cur_similar_word) for cur_similar_word
    return ret_list

"""
for example:
sentence_augmentation("I want to find a beautiful place to go", w2v_model)

result: (all this sentences share the same label with the original sentence)
['we want to find a beautiful place to go',
 'you want to find a beautiful place to go',
 'I wanted to find a beautiful place to go',
 'I need to find a beautiful place to go',
 'I prefer to find a beautiful place to go',
 'I wanna to find a beautiful place to go',
 'I want to discover a beautiful place to go',
 'I want to locate a beautiful place to go',
 'I want to uncover a beautiful place to go',
 'I want to find a gorgeous place to go',
 'I want to find a lovely place to go',
 'I want to find a stunningly_beautiful place to go',
 'I want to find a breathtakingly_beautiful place to go',
 'I want to find a wonderful place to go',
 'I want to find a fabulous place to go',
 'I want to find a loveliest place to go',
 'I want to find a beatiful place to go',
 'I want to find a magnificent place to go',
 'I want to find a beautiful palce to go',
 'I want to find a beautiful places to go',
 'I want to find a beautiful spot to go',
 'I want to find a beautiful place to sit',
 'I want to find a beautiful place to stay']
 """
```

Even though there are some grammar errors, machine does not care. So basically, we can use the augmented data.

Data's problem is done. Now we can use the dataset to train a SVM.

Step 4 of our project --- recommendation

In this project, we use cosine similarity as reference when recommend content to users.

Firstly, we use the following function (word2vec method) to generate features for each toplist, destination, attraction

```
def generate_w2v_feature(words, w2v_model, feature_size=300):
    return np.mean([w2v_model[w] for w in nltk.word_tokenize(words) if w in w2v_model]
                   or [np.zeros(feature_size)], axis=0)
```

After we get the feature of our dataset (toplists, destinations, attractions), we generate the feature of users' input by the above function. Then we calculate the cosine similarities and return the id of the most similar one.

```
def find_most_similar_toplist(toplist_df, sentence, w2v_model, threshold=0.3):
    x = generate_w2v_feature(sentence, w2v_model).reshape(1, -1)
    similarity_with_toplist = np.array(
        [cosine_similarity(x, cur_toplist_feature.reshape(1, -1))
         for cur_toplist_feature in toplist_df["feature"]]
    ).reshape((-1, ))
    idx = similarity_with_toplist.argmax()
    return toplist_df["id"][idx] if similarity_with_toplist[idx] > threshold else -1
```

Then, we can find the record in our dataset by id and recommend it to our users.

```
def recommend_toplist_by_sentence(toplist_df, sentence, w2v_model, db_connector):
    toplist_id = find_most_similar_toplist(toplist_df, sentence, w2v_model)
    if toplist_id == -1:
        return None
    mycursor = db_connector.cursor(buffered=True)
    mycursor.execute("SELECT * FROM toplist WHERE id=%d" % toplist_id)
    toplist = mycursor.fetchone()

    return {"id": toplist[0], "name": toplist[1], "link": toplist[2]}
```

Step 4 of our project --- deployment using docker container and docker-compose