Моя курсова робота — це проект бота для автоматичної відповіді на запитання.

Призначення та коротка характеристика

Цей бот призначений для надання швидкої відповіді на поставлене запитання у Slack каналі programming_2016_2017(не виключається застосування цієї програми в інших ресурсах наприклад на форумах запитань). Відповідь надається лише в тому випадку, якщо це запитання уже ставилося, тобто воно є у базі запитань-відповідей(важливим є що запитанням вважається стрічка яка завершується знаком питання. В подальшому буде розроблено більш ґрунтовний аналіз типу речення). У разі відсутності відповіді на поставлене запитання воно переадресовується користувачам, які володіють необхідною інформацією(в даному випадку викладачам). Відповідь від такої особи запам'ятовується і запитання разом із відповіддю на нього додається до бази запитань-відповідей. Якщо отримане повідомлення не є ні запитанням, ні відповіддю на запитання воно ігнорується.

Увесь процес відбувається наступним чином.

Підєднання та автентифікація.

Після запуску бот підєднується до Slack сервера, на якому розташована команда Slack, яка додала цього бота. Якщо зєднання успішне, тобто наявне зєднання інтернетом та токен бота коректний, бот починає функціювати. Інакше закінчується робота програми.

Створення контейнерів для зберігання даних

Після запуску створюється обєкт власного абстрактного типу даних MyMultiset, який містить попередньо отримані запитання-відповіді та обєкт власного типу даних для зберігання запитань, які очікують на відповідь.

Обробка вхідних подій

Кожне зчитування та обробка повідомлень відбуваються з інтервалом одна секунда.

1) **Зчитування.** Після підєднання бот за допомогою Slack Real Time Messaging (RTM) API через веб-сокети зчитує останні події(в тому числі повідомлення) кожного Slack-каналу (до якого його запросили) і зберігає їх у список.
2) **Виокремлення тексту повідомлення.** Кожна подія обробляється почергово. Подія представлена словником. За ключами цього словника отримується ID користувача, який запостив повідомлення, текст повідомлення, Slack канал в якому це повідомлення було відправлено.
3) **Визначення типу речення.** Повідомлення може бути трьох типів: запитання, відповідь на запитання чи просто звичайне повідомлення.
4) **Обробка речення в залежності від типу.**

    4.1) **Запитання.** Речення вважається запитанням, якщо воно в кінці містить знак запитання. Отримане запитання порівнюється із запитаннями із бази запитань-відповідей за допомогою бібліотеки gensim. Якщо коефіцієнт подібності найбільш схожого запитання до заданого вищий за 0.99, тоді по вертається відповідь на нього із бази запитань-відповідей і бот постить її в канал. Якщо ж найбільш схоже запитання має менший коефіцієнт подібності бот постить задане запитання у канал із зверненням до викладачів(@teacher_name) і додає це запитання контейнеру запитань, які очікують на відповідь.

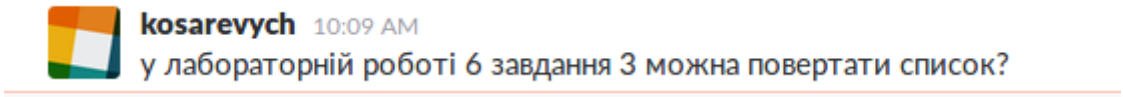
4.2) **Відповідь.** Речення вважається відповіддю, якщо у ньому контейнер запитань, які очікують на відповідь не порожній і у реченні є звертання до користувача, який поставив запитання(@username). Запитання на яке було дано відповідь вилучається із контейнеру запитань без відповідей. Запитання разом із отриманою відповіддю на нього додається до бази запитань-відповідей. Таким чином база постійно поповнюється новими запитаннями.

4.3) **Звичайне речення.** Речення вважається звичайним, якщо воно не є ані запитанням, ані відповіддю. Таке речення ігнорується ботом.

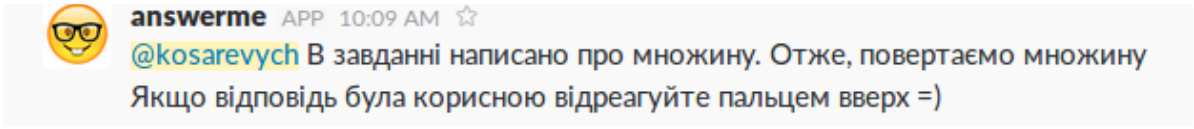
Після обробки речення та виведення відповіді бота описаний процес починається з початку.

Вхідні та вихідні дані програми

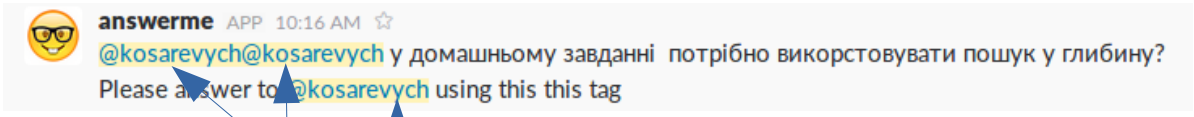
1) **Вхідні дані:**
   - Історія повідомлень у Slack каналі programming_2016_2017 на основі яких формується база запитань-відповідей.
   - Повідомлення надіслане у Slack канал до якого додано бота.



2) **Вихідні дані:**
   - відповідь на запитання

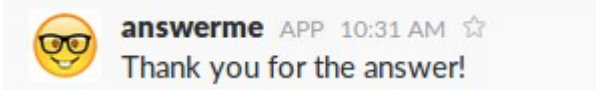


   - переадресація запитання до викладачів



(тут перші два звертання – імена викладачів, так як це канал для тестування у ньому тільки 1 користувач; звертання у другому рядку ім'я користувача, який поставив запитання)

   - подяка за надану відповідь



Структура програми

- Slackbot/ # коренева директорія
  - docs/ # текстові файли проекту
    - Описи етапів/
    - answers_base.txt # відповіді на запитання
    - origin_messages.txt # не змінені повідомлення
    - messages.txt # змінені повідомлення для полегшення обробки
    - ukrainian-stopwords.txt # стоп-слова(шумові слова) української мови
  - modules/
    - message_processing/ # пакет для роботи з повідомленнями
      - get_messages.py # зчитування історії повідомлень каналу
      - get_questions.py # виокремлення запитань та відповідей
      - main.py # для обробки отриманого повідомлення
    - my_multiset/ # пакунок для Mymultiset ADT
      - tmp/ # для зберігання тимчасових даних при визначенні подібності запитань
      - arrays.py # містить структуру даних Array
      - my_corpus.py # містить клас для створення корпусу запитань

        my_multiset.py # містить клас даного ADT
      - question.py # містить клас для зберігання запитання та відповіді на нього
      - str_indx.py # модуль для перевірки функціоналу бібліотеки gensim, який використовується в даному ADT)
      - tst_multiset.py # модуль для перевірки працездатності ADT
    - questions_dict/ # пакунок для questions_dict.py
      - questions_dict.py # містить клас для зберігання запитань, які очікують на відповідь
    - get_user_id.py # визначення id користувача Slack
  - run.py # запуск бота
  - setup.py

1) modules/message_processing/get_questions.py

   1.1) Функції:

   - get_questions(): створює пари запитання-відповідь, повертає список пар

   Також містить функції для отримання, вилучення адресата в повідомленні(@username)

2) modules/message_processing/main.py

   2.1. Функції:

   - main(): визначення типу речення, виклик відповідних функцій в залежності від типу речення

- create_multiset(): створює обєкт типу MyMultiset із запитаннями на які є відповідь
- is_question(*message*): визначає чи *message* є запитанням
- is_answer(*multiset*, *non_answ_dict*, *message*): перевіряє чи message є відповіддю на запитання на основі аналізу *message* на наявність звертання до користувача, який поставив питання та *non_answ_dict* на наявність такого такого користувача. Якщо ці умови виконуються із *non_answ_dict* отримується запитання(обєкт класу Question, який може містити запитання і відповідь), цей обєкт вилучається; до цього обєкту додається відповідь(початково було тільки запитання) – *message*. Обєкт класу Question із запитанням та відповідю додається до *multiset.*

3) Modules/my_multiset/my_corpus.py

3.1. Класи:

3.1.1) MyCorpus(): **Атрибути**: dictionary(обєкт класу Dictionary gensim бібліотеки), texts(адреса файлу із запитаннями). **Методи**:__iter__: зчитує файл із запитаннями. Перетворює кожне із них у вектор за допомогою функції doc2bow() із бібліотеки gensim. Повертає ці вектори у формі генератора, що значно змешує обєм використовуваної оперативної памяті, тому що одночасно в памяті знаходиться тільки один вектор, а не всі.

4) Modules/my_multiset/question.py

4.1. Клас Question:

4.1.1) Атрибути:

- _question: запитання, обовязковий атрибут
- _value: відповідь на запитання, необовязковий атрибут
- _user: ID користувача, який поставив запитання, необовязковий атрибут

4.1.2) Методи:

- set_value(self, value): встановлює атрибут _value у значення *value*
- set_user_to_None: видаляє значення користувача
- Решта методів призначені для отримання значень атрибутів

5) modules/questions_dict/questions_dict.py

5.1. Клас Quetions_dict:

5.1.1) Атрибути:

- _dict: Python словник, ключ: ID користувача, який поставив питання, значення_ключа: обєкт класу Question

5.1.2) Методи:

- is_non_answered(self): перевіряє чи є запитання, які чекають на відповідь
- add_question(question): додає запитання до словника за ключем, який отримується із *question*(обєкт класу Question)
- get_quest_to_answer(self, user): повертає ПЕРШЕ запитання із списку запитань за ключем *user*

- remove_question(self, user): вилучає запитання із списку за ключем *user*, якщо це запитання єдине – вилучає ключ *user*
- get_users(self): повертає список ключів словника(користувачів)

6) run.py

6.1. Функції:

- parse_slack_output(*slack_rtm_output*): із кожного повідомлення(словник) із *slack_rtm_output(список словників)* за ключами “user”, “text”, “channel” отримує їхні значення.
- Handle_message(): викликає функцію main із message_processing/main.py. На основі її відповіді постить відповідне повідомлення в Slack канал.(Приклади таких повідомлень містяться в пункті Вхідні та вихідні дані програми)

Інструкція по користуванню програмою

Запуск бота: python run.py

У каналі Slack:

Як ставитити запитання: вкінці речення ОБОВЯЗКОВО поставити знак запитання.

Як відповідати: у відповіді важливо звернутися до того хто поставив запитання. Наприклад: “@username YOUR ANSWER”. Відповідати може будь-який користувач. ВАЖЛИВО! На два і більше питань одного користувача слід відповідати у тому порядку в якому вони були задані.

Тестові приклади

Тестування правильності відповідей бота проводилось в каналі спеціально створеної Slack команди вручну, тому що неможливо автоматично перевірити правильність отриманої відповіді.

Тесткейси:

- відоме боту питання в різних інтерпретаціях
- невідоме питання для бота
- відповідь на невідоме запитання