Московский авиационный институт
Факультет прикладной математики и физики

Курсовой проект

по курсу:
«Информационный поиск»
по теме:
«Диалоговый чат-бот»
2 семестр

Студент: Ахмед С. Х.
Преподаватель: Калинин А. Л.
Группа: 8О-106М

Москва, 2019 г

Постановка задачи

Спроектировать и реализовать на любом выбранном языке программирования Диалогового Чат-бота. Объяснить алгоритм построения и недостатки выбранного подхода. С какими трудностями столкнулись? Привести примеры его работы. Протестировать его работу.

Ход решения

Conversational UI (user interfaces) – это разговорные или диалоговые интерфейсы, рассчитанные на взаимодействие с пользователем. На таких интерфейсах в свою очередь строятся различные медиа и сервисы. С их помощью грань между медиа и сервисами всё чаще стирается, предоставляя пользователю не только полезную информацию, но и решения по разным вопросам.

В качестве основных подходов к реализации чат-ботов, я рассматривал следующие:

1) Реализация чат-бота на основе seq2seq моделей (с вниманием) нейронных сетей (Пример подобной реализации можно посмотреть на https://neuralconvo.huggingface.co/)
2) Реализация на основе алгоритмов поиска документов (с использованием и нет базы данных)

Но, прежде чем затронуть тему с выбранными путями, стоит проговорить про первую проблему, связанную с реализацией. Хороших и объемных датасетов диалогов на русском языке к сожалению мало(лично я их мало нашел, поэтому пришлось формировать свой), помимо прочего широко распространённые диалоги на английском языке (я брал ubuntu dialogs, cornell movie dataset и twitter) имеют свои ограничения, а также очень много спец символов разметки(это особенно касается ubuntu dialogs(тематические обсуждения)). Помимо этих корпусов, я собрал еще один содержащий рандомные фрагменты диалогов:

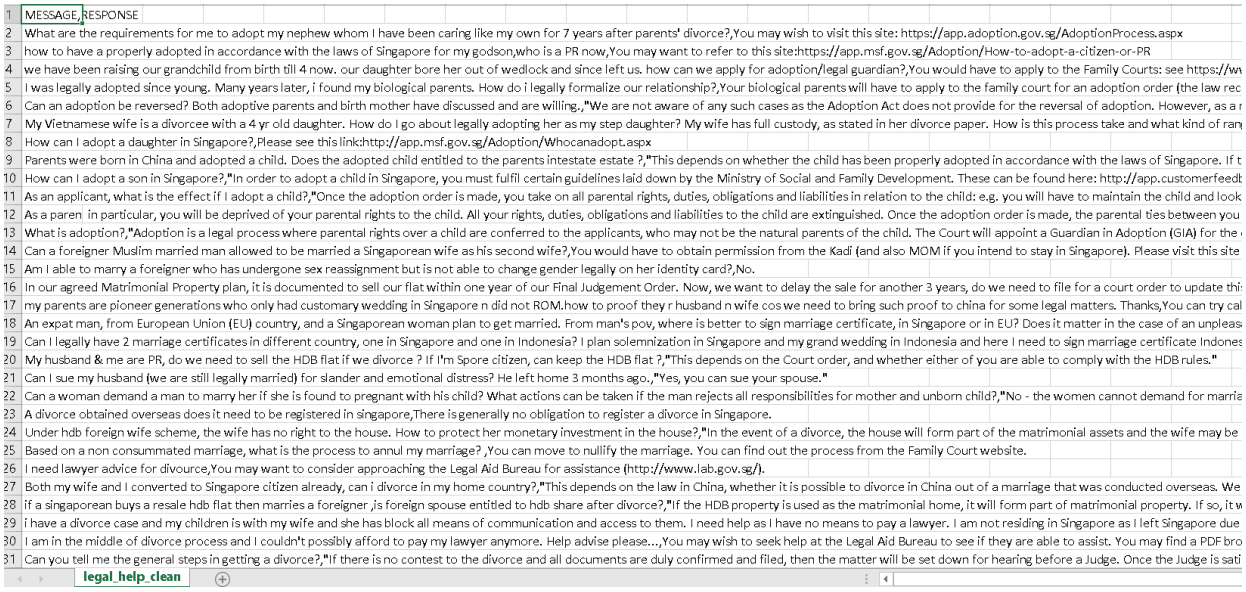


Рис. 1 Собранные диалоги

	Context	Utterance	Label
0	i think we could import the old comment via rsync , but from there we need to go via email . i think it be easier than cach the status on each bug and than import bite here and there __eou__ __eot__ it would be veri easi to keep a hash db of message-id __eou__ sound good __eou__ __eot__ ok __eou__ perhap we can ship an ad-hoc apt_preferec __eou__ __eot__ version ? __eou__ __eot__ thank __eou__ __eot__ not yet __eou__ it be cover by your insur ? __eou__ __eot__ yes __eou__ but it 's realli no...	basic each xfree86 upload will not forc user to upgrad 100mb of font for noth __eou__ no someth i do in my spare time . __eou__	1
1	i 'm not suggest all - onli the one you modifi . __eou__ __eot__ ok , it sound like you re agre with me , then __eou__ though rather than `` the one we modifi '' , my idea be `` the one we need to merg '' __eou__ __eot__	sorri __eou__ i think it be ubuntu relat . __eou__	0
2	afternoon all __eou__ not entir relat to warti , but if grub-instal take 5 minut to instal , be this a sign that i should just retri the instal :) __eou__ __eot__ here __eou__ __eot__ you might want to know that thinic in warti be buggi compar to that in sid __eou__ __eot__ and appar gnome be suddent almost perfect (out of the thinic problem) , nobodi report bug : -p __eou__ i do n't get your question , where do you want to past ? __eou__ __eot__ can i file the panel not link to ed ? :) ...	yep . __eou__ oh , okay . i wonder what happen to you __eou__ what distro do you need ? __eou__ yes __eou__	0
3	interest __eou__ grub-instal work with / be ext3 , fail when it be xfs __eou__ i think d-i instal the relev kernel for your machin . i have a p4 and it instal the 386 kernel __eou__ holi crap a lot of stuff get instal by default :) __eou__ you be instal vim on a box of mine __eou__ ;) __eou__ __eot__ more like osx than debian ;) __eou__ we have a select of python modul avail for great justic (and python develop) __eou__ __eot__ 2.8 be fix them iirc __eou__ __eot__ pong __eou__ vino will...	that the one __eou__	1
4	and becaus python give mark a woodi __eou__ __eot__ i 'm not sure if we re mean to talk about that public yet . __eou__ __eot__ and i think we be a `` pant off '' kind of compani ... : p __eou__ you need new glass __eou__ __eot__ mono 1.0 ? dude , that 's go to be a barrel of laugh for total non-releas relat reason dure hoari __eou__ read bryan clark 's entri about networkmanag ? __eou__ __eot__ there be an accompani irc convers to that one < g > __eou__ explain ? __eou__ i guess you could s...	(i think someon be go to make a joke about .au bandwidth ...) __eou__ especi not if you re use screen ;) __eou__	1

Рис. 2 Ubuntu dialogs

Как видно, во-первых, в тексте(это ubuntu dialogs) очень много знаков разметки, во-вторых, сам язык наполнен грамматическими ошибками обусловленная лемматизацией, в-третьих, он узко специализирован, что не дает возможности, поговорить о других темах. Поэтому я принял решение собрать побольше диалогов (Рис. 1).

Проблема корпуса Twitter – он нацелен на короткие сообщения, что также не является хорошим признаком. Но у него широкая вариабельность в темах, что нивелирует проблему данного корпуса. (некоторые скрипты для получения данных представлены в приложении).

Посмотрим на распределение слов(длин слов)

Выбор метода решения

Как упоминалось выше, я рассматривал два варианта решения задачи.

Нейросетевое решение – решение на основе модели нейронных сетей порождающие цепочки символов на основе входной цепочки(seq2seq). Обычно в такую сеть входят блоки LSTM(Long-Short Term Memory) и GRU ячейки. Основными проблемами такого подхода являются:

1) Необходимы ресурсы для обучения подобных сетей, особенно ценится наличие графических процессоров (GPU). Также ресурсы потребуются на этапе прогона модели (особенно это касается Tensorflow, он любит сжевывать память, а ограничение в 66% не сильно срабатывает)
2) Время отклика очень велико (касается запуска на ноутбуках и обычных ПК, в продакшене реализуется посредством серверов или облачной инфраструктуры, поэтому такой проблемы нет).

По этим двум причинам я решил отказаться он упора на этот подход. Несмотря на минусы точность такой модели может быть достаточно велика. Пример работы:

```
replies = []
for ii, oi in zip(input_.T, output):
    q = data_utils.decode(sequence=ii, lookup=metadata['idx2w'], separator=' ')
    decoded = data_utils.decode(sequence=oi, lookup=metadata['idx2w'], separator=' ').split(' ')
    if decoded.count('unk') == 0:
        if decoded not in replies:
            print('q : [{0}]; a : [{1}]'.format(q, ' '.join(decoded)))
            replies.append(decoded)
```

```
q : [go see them]; a : [i was thinking of that]
q : [a unk of the before we say goodnight from last at unk state park in maine]; a : [this is a great idea to be a great day fo
r a while]
q : [unless you want this ]; a : [what is he]
q : [my fav moment from the debate last night]; a : [wait for the first time]
q : [ok i dont see any catholic terrorist]; a : [not even to be a threat to the point]
q : [thats your sister]; a : [i dont know what i was talking about]
q : [radical islam unk a threat to the united states and our leaders need to realize this before its too late]; a : [is that a
joke]
q : [literally the same thing happened to me ]; a : [thats what i said]
q : [last chance to win vip passes to new york comic con amp meet the walking dead cast go]; a : [if you were in the same place
i dont have to see it]
q : [4 correction that video is from a few years ago]; a : [its been a fan of my life but i dont know what to do]
q : [we are doing band there are many unk in the band]; a : [i have to be a good job]
q : [im sure it does but i doubt ill see a unk amount of money from it]; a : [i think its a good idea]
q : [which report did this graph come from the unk unk data is really interesting and hard to find]; a : [is this a lot of time
s from the world of the world of the year]
q : [thats pretty cool but now i am dying to find out what s first tweet will be]; a : [it was a good time for the first time]
q : [great debate poll numbers  i will be on at 700 to discuss enjoy]; a : [dont have to be a president]
q : [yooo me too]; a : [ahh i know i was just thinking about this]
q : [the ladies of could not be more idiotic or blind they should invite on for a talk]; a : [they are not sarcastic]
q : [clothes that dont fit me no more]; a : [ur so cute]
q : [good evening unk]; a : [good morning all]
q : [so pretty much she didnt want him so u were is 2nd choice ]; a : [lol i dont know what to say about it ]
q : [another reminder of my favorite quote people sell what they need and buy what they dont need thanks]; a : [this is a joke]
q : [the point is there is unk to a hate and fear unk campaign]; a : [you dont care about the debate that is not a fact that yo
u are not a racist]
q : [which one is he]; a : [hes a good one]
```

В качестве основной модели были выбраны подходы с поиском. Первоначально создается база данных в которую заносятся фрагменты диалогов со сторон общающихся, по которой дальше происходит поиск с помощью 2 метрик: Расстояние Левенштейна, коэффицент Жаккара. Поиск происходит по пространству текстов участников диалогов, выбирается оптимальный фрагмент диалога и выдается соответствующая часть ответа. Естественно, тексты предварительно очищены.

```
import SequenceMatcher
```

```
class Comparator:

    def __init__(self, language):

        self.language = language

    def __call__(self, statement_a, statement_b):
        return self.compare(statement_a, statement_b)

    def compare(self, statement_a, statement_b):
        return 0


class LevenshteinDistance(Comparator):
    def compare(self, statement_a, statement_b):
        # Return 0 if either statement has a falsy text value
        if not statement_a.text or not statement_b.text:
            return 0

        # Get the lowercase version of both strings
        statement_a_text = str(statement_a.text.lower())
        statement_b_text = str(statement_b.text.lower())

        similarity = SequenceMatcher(
            None,
            statement_a_text,
            statement_b_text
        )

        # Calculate a decimal percent of the similarity
        percent = round(similarity.ratio(), 2)

        return percent
class JaccardSimilarity(Comparator):
    def __init__(self, language):
        super().__init__(language)
        import spacy

        self.nlp = spacy.load(self.language.ISO_639_1)

    def compare(self, statement_a, statement_b):
        # Make both strings lowercase
        document_a = self.nlp(statement_a.text.lower())
        document_b = self.nlp(statement_b.text.lower())

        statement_a_lemmas = set([
            token.lemma_ for token in document_a if not token.is_stop
        ])
        statement_b_lemmas = set([
```

```
            token.lemma_ for token in document_b if not token.is_stop
        ])

        # Calculate Jaccard similarity
        numerator = len(statement_a_lemmas.intersection(statement_b_lemmas))
        denominator = float(len(statement_a_lemmas.union(statement_b_lemmas)))
        ratio = numerator / denominator

        return ratio
```

Иными словами мы итерируемся по базе знаний в поисках наиболее совпадающего нам тэга или фрагмента диалога (по указанным метрикам) и вытаскиваем оттуда соответствующий ответ.

```
class BestMatch():
    def __init__(self, chatbot, **kwargs):
        super().__init__(chatbot, **kwargs)

        self.excluded_words = kwargs.get('excluded_words')

    def process(self, input_statement,
additional_response_selection_parameters=None):
        search_results = self.search_algorithm.search(input_statement)
        closest_match = next(search_results, input_statement)

        # Search for the closest match to the input statement
        for result in search_results:
            closest_match = result

            # Stop searching if a match that is close enough is found
            if result.confidence >= self.maximum_similarity_threshold:
                break

        self.chatbot.logger.info('Using "{}" as a close match to "{}" with a
confidence of {}'.format(
            closest_match.text, input_statement.text, closest_match.confidence
        ))

        recent_repeated_responses = filters.get_recent_repeated_responses(
            self.chatbot,
            input_statement.conversation
        )

        for index, recent_repeated_response in
enumerate(recent_repeated_responses):
            self.chatbot.logger.info('{}. Excluding recent repeated response of
"{}"'.format(
                index, recent_repeated_response
            ))
```

```
        response_selection_parameters = {
            'search_in_response_to': closest_match.search_text,
            'exclude_text': recent_repeated_responses,
            'exclude_text_words': self.excluded_words
        }

        alternate_response_selection_parameters = {
            'search_in_response_to':
self.chatbot.storage.tagger.get_text_index_string(
                input_statement.text
            ),
            'exclude_text': recent_repeated_responses,
            'exclude_text_words': self.excluded_words
        }

        if additional_response_selection_parameters:

response_selection_parameters.update(additional_response_selection_parameters)

alternate_response_selection_parameters.update(additional_response_selection_para
meters)
        response_list =
list(self.chatbot.storage.filter(**response_selection_parameters))

        alternate_response_list = []

        if not response_list:
            self.chatbot.logger.info('No responses found. Generating alternate
response list.')
            alternate_response_list =
list(self.chatbot.storage.filter(**alternate_response_selection_parameters))

        if response_list:
            self.chatbot.logger.info(
                'Selecting response from {} optimal responses.'.format(
                    len(response_list)
                )
            )

            response = self.select_response(
                input_statement,
                response_list,
                self.chatbot.storage
            )

            response.confidence = closest_match.confidence
            self.chatbot.logger.info('Response selected. Using
"{}"'.format(response.text))
        elif alternate_response_list:
            self.chatbot.logger.info(
                'Selecting response from {} optimal alternate responses.'.format(
```

```
                    len(alternate_response_list)
                )
            )
            response = self.select_response(
                input_statement,
                alternate_response_list,
                self.chatbot.storage
            )

            response.confidence = closest_match.confidence
            self.chatbot.logger.info('Alternate response selected. Using
"{}"'.format(response.text))
        else:
            response = self.get_default_response(input_statement)

        return response
```

```
def get_most_frequent_response(input_statement, response_list, storage=None):
    matching_response = None
    occurrence_count = -1

    logger = logging.getLogger(__name__)
    logger.info('Selecting response with greatest number of occurrences.')

    for statement in response_list:
        count = len(list(storage.filter(
            text=statement.text,
            in_response_to=input_statement.text)
        ))

        # Keep the more common statement
        if count >= occurrence_count:
            matching_response = statement
            occurrence_count = count

    # Choose the most commonly occuring matching response
    return matching_response


def get_first_response(input_statement, response_list, storage=None):
    logger = logging.getLogger(__name__)
    logger.info('Selecting first response from list of {} options.'.format(
        len(response_list)
    ))
    return response_list[0]


def get_random_response(input_statement, response_list, storage=None):
    from random import choice
```

```
    logger = logging.getLogger(__name__)
    logger.info('Selecting a response from list of {} options.'.format(
        len(response_list)
    ))
    return choice(response_list)
```

И последний подход с TF-IDF и косинусным расстоянием между векторами слов, основан на идее поиска: Мы ищем в пространстве документов наиболее релевантные документы(релевантность оценивается с помощью TF-IDF), вычисляется косинусное расстояние с нашим исходним сообщением, и смотрятся пороги по сходству: если наша мера меньше 0.31, то считается, что наше сообщение не было распознано, иначе из вектора ответов, соответствующему наиболее релевантному документу, рандомным образом выбирается ответ(в случае если процент совпадения больше 90 процентов иначе берется конкретный ответ) выбирается ответ и выдается пользователю. Оценка для этого подхода:

```
Recall @ (1, 10): 0.495032
Recall @ (2, 10): 0.596882
Recall @ (5, 10): 0.766121
Recall @ (10, 10): 1
```

Сравнивая с рандомным выбором, имеем:

```
Recall @ (1, 10): 0.0967759
Recall @ (2, 10): 0.199524
Recall @ (5, 10): 0.499101
Recall @ (10, 10): 1
```

Соответствующий код:

```
sentences = []
    Tokens = RemovePunctuation(nltk.word_tokenize(content))
    pos_tokens = [word for word,pos in pos_tag(Tokens, tagset='universal')]
    word_tokens = LemTokens(pos_tokens)
    filtered_sentence = []
    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)
    filtered_sentence =" ".join(filtered_sentence).lower()
    test_set = (filtered_sentence, "")
    try:
        with open(tfidf_vectorizer_pickle_path, 'rb') as f:
            tfidf_vectorizer = pickle.load(f)
        with open(tfidf_matrix_pickle_path, 'rb') as f:
            tfidf_matrix_train = pickle.load(f)
    except:
        start = timeit.default_timer()
        with open(json_file_path) as sentences_file:
            reader = json.load(sentences_file)
            for row in reader:
                db_tokens = RemovePunctuation(nltk.word_tokenize(row['MESSAGE']))
```

```
                pos_db_tokens = [word for word,pos in pos_tag(db_tokens,
tagset='universal')]
                db_word_tokens = LemTokens(pos_db_tokens)
                db_filtered_sentence = []
                for dbw in db_word_tokens:
                    if dbw not in stop_words:
                        db_filtered_sentence.append(dbw)
                db_filtered_sentence =" ".join(db_filtered_sentence).lower()
                sentences.append(db_filtered_sentence)
                i +=1
            tfidf_vectorizer = TfidfVectorizer()
            tfidf_matrix_train = tfidf_vectorizer.fit_transform(sentences)
            stop = timeit.default_timer()
            print ("Training Time : ")
            print (stop - start)
            with open(tfidf_vectorizer_pickle_path, 'wb') as f:
                pickle.dump(tfidf_vectorizer, f)

            with open(tfidf_matrix_pickle_path, 'wb') as f:
                pickle.dump(tfidf_matrix_train, f)
    tfidf_matrix_test = tfidf_vectorizer.transform(test_set)
    cosine = cosine_similarity(tfidf_matrix_test, tfidf_matrix_train)
    idx= cosine.argsort()[0][-2]
    flat =  cosine.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if (req_tfidf==0): #Threshold A
        not_understood = "Apology, I do not understand. Can you rephrase?"
        return not_understood
    else:
        cosine = np.delete(cosine, 0)
        max = cosine.max()
        response_index = 0
        if (max <= 0.34): #Threshold B
            not_understood = "Apology, I do not understand. Can you rephrase?"
            return not_understood
        else:
                if (max > 0.91): #Threshold C
                    new_max = max - 0.05
                    list = np.where(cosine > new_max)
                    response_index = random.choice(list[0])
                else:
                    response_index = np.where(cosine == max)[0][0] + 2
                j = 0
                with open(json_file_path, "r") as sentences_file:
                    reader = json.load(sentences_file)
                    for row in reader:
                        j += 1
                        if j == response_index:
                            return row["RESPONSE"]
```

Платформа самого чат-бота

В качестве альтернатив рассматривались следующие варианты платформ:

1) TelegramAPI
2) DiscordAPI
3) MS Bot Framework

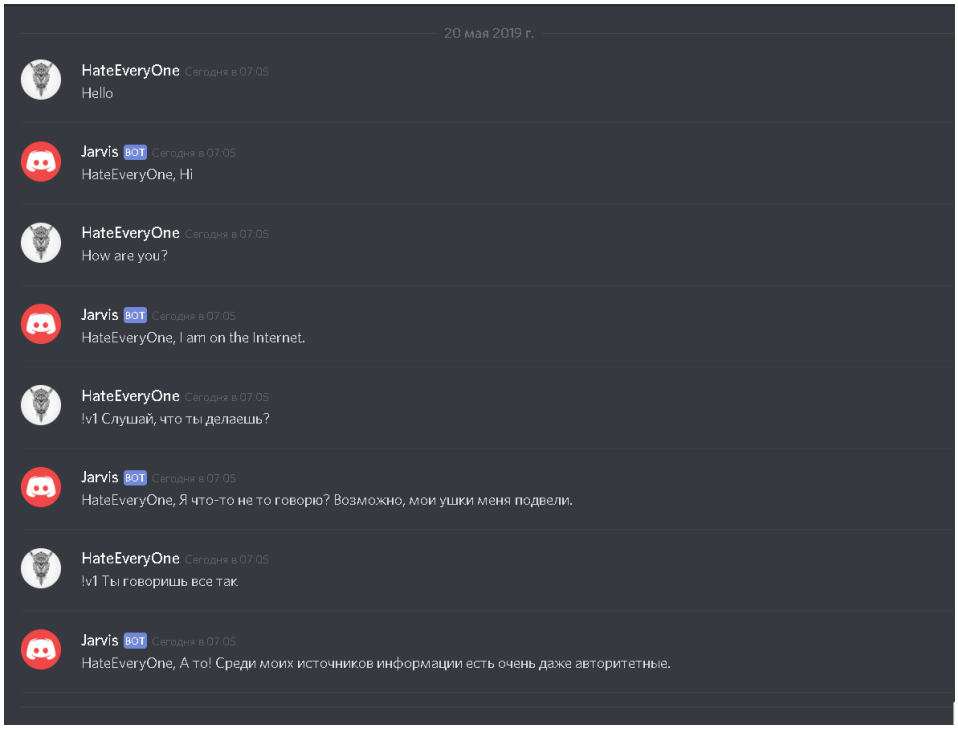
Проблема TelegramAPI заключается в том, что из-за проблем с Роскомнадзором(Telegram блокируется на территории России), напрямую общаться с сервером telegram весьма проблематично. Эту проблему можно решить подняв VPN или сервак в облаке и адресовать запросы оттуда, но это слишком финансово затратно. Также это можно решить с помощью Bot Framework, но опять необходим доступ к облаку, также необходимо использовать язык C#. Это не является проблемой, однако все модули реализованы на языке Python, перекладывание на использование с C# потребует развертывания бессерверной инфраструктуры(Azure Functions).Есть конечно коннекторы для BF под Python, однако они вышли недавно и все еще сильно сырые.

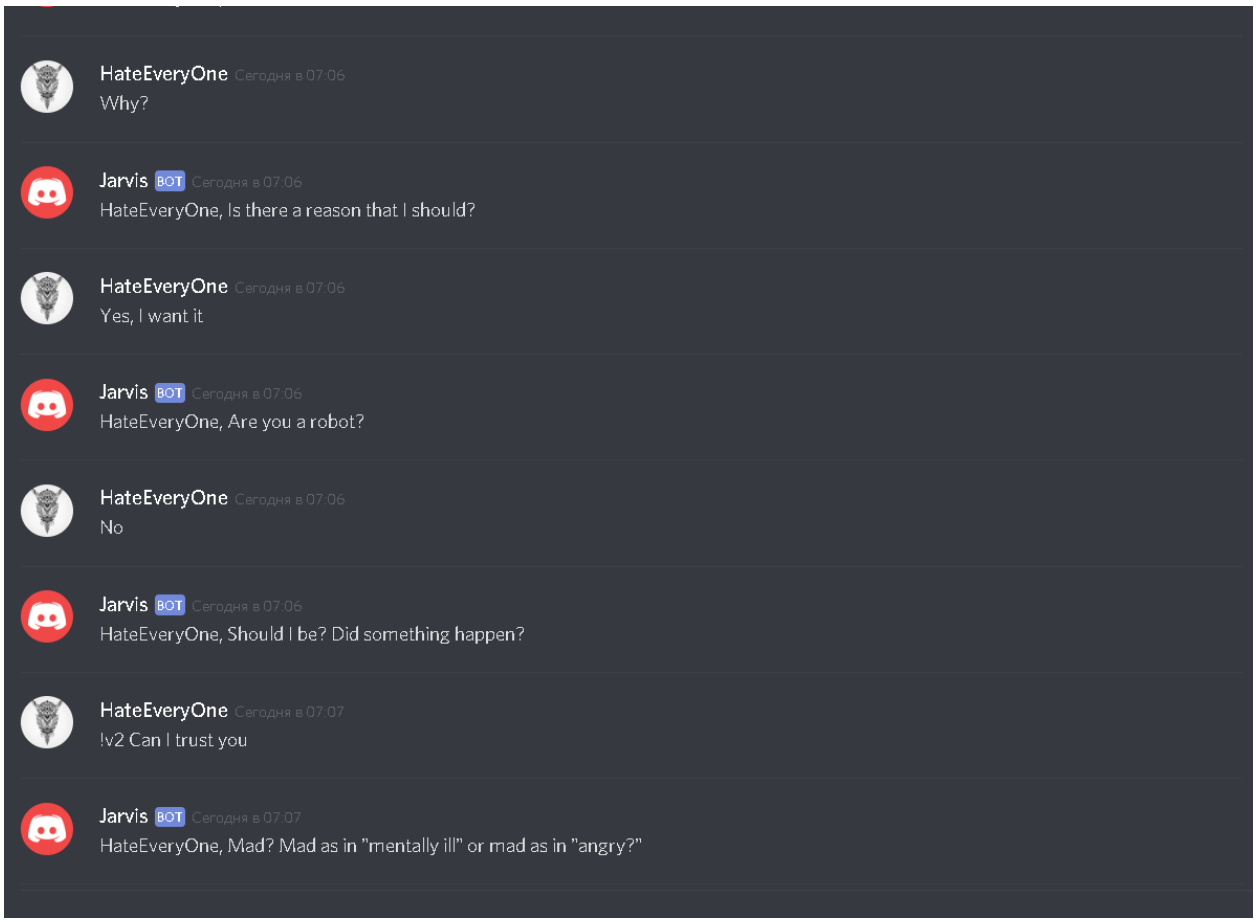
Поэтому мой выбор остановился на Discord. Там хорошо документированное API, хорошая обертка под всевозможные службы и поддержка деплоя собственных ботов.

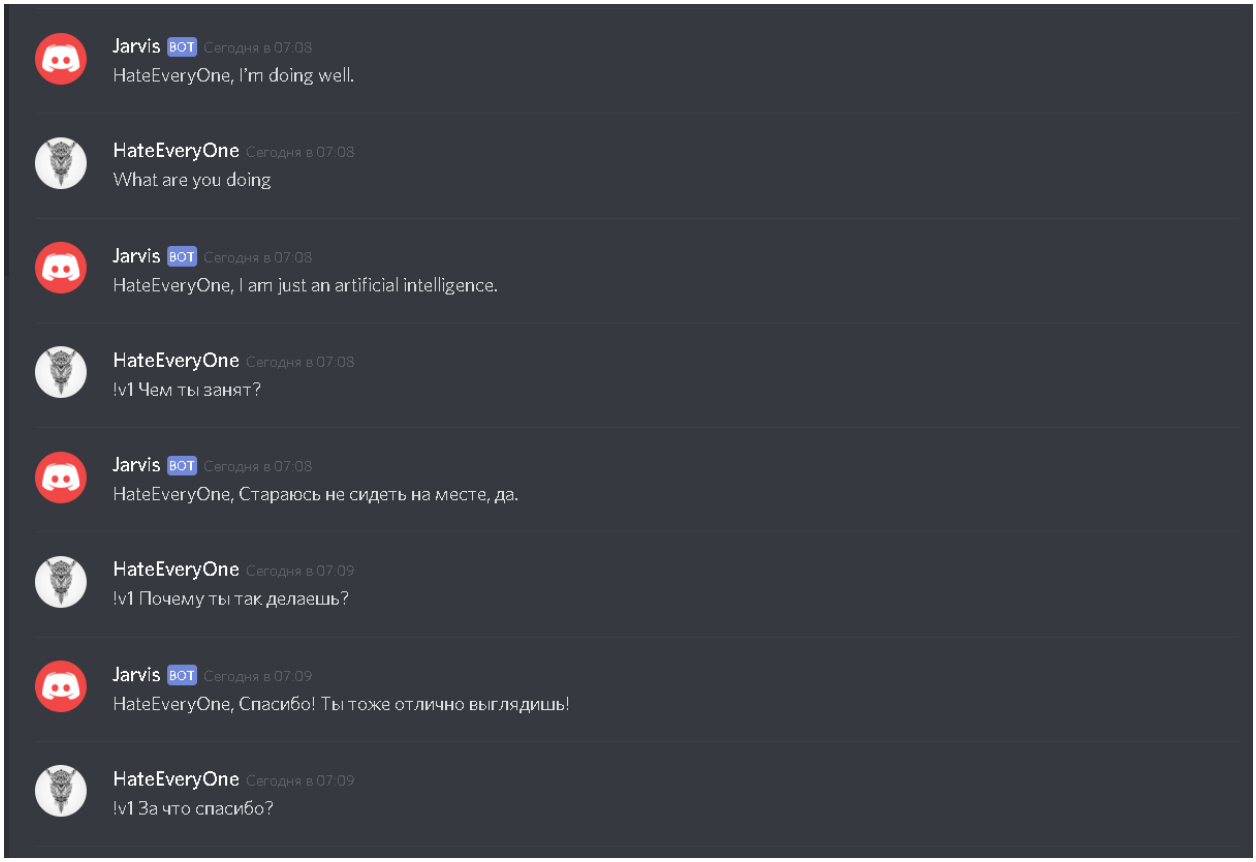
Скриншоты

Команда !v1 – поддержку русскоязычного датасета, в базовом варианте используется английский язык для общения(причины были объяснены выше). Видео демонстрация работы бота доступна

Ссылке: https://drive.google.com/drive/folders/1_29iIvMJBzOsGj1seG93qMN35Bj2W0Ni?usp=sharing




HateEveryOne Сегодня в 07:06
Why?
Jarvis BOT Сегодня в 07:06
HateEveryOne, Is there a reason that I should?
HateEveryOne Сегодня в 07:06
Yes, I want it
Jarvis BOT Сегодня в 07:06
HateEveryOne, Are you a robot?
HateEveryOne Сегодня в 07:06
No
Jarvis BOT Сегодня в 07:06
HateEveryOne, Should I be? Did something happen?
HateEveryOne Сегодня в 07:07
!v2 Can I trust you
Jarvis BOT Сегодня в 07:07
HateEveryOne, Mad? Mad as in "mentally ill" or mad as in "angry?"


Jarvis BOT Сегодня в 07:08
HateEveryOne, I'm doing well.
HateEveryOne Сегодня в 07:08
What are you doing
Jarvis BOT Сегодня в 07:08
HateEveryOne, I am just an artificial intelligence.
HateEveryOne Сегодня в 07:08
!v1 Чем ты занят?
Jarvis BOT Сегодня в 07:08
HateEveryOne, Стараюсь не сидеть на месте, да.
HateEveryOne Сегодня в 07:09
!v1 Почему ты так делаешь?
Jarvis BOT Сегодня в 07:09
HateEveryOne, Спасибо! Ты тоже отлично выглядишь!
HateEveryOne Сегодня в 07:09
!v1 За что спасибо?

Скрипты, с помощью которого я добывал некоторые корпусы документов

```
def translate_dialog_to_lists(dialog_filename):
    """
    Translates the dialog to a list of lists of utterances. In the first
    list each item holds subsequent utterances from the same user. The second
level
    list holds the individual utterances.
    :param dialog_filename:
    :return:
    """

    dialog_file = open(dialog_filename, 'r')
    dialog_reader = unicodecsv.reader(dialog_file,
delimiter='\t',quoting=csv.QUOTE_NONE)

    # go through the dialog
    first_turn = True
    dialog = []
    same_user_utterances = []
    #last_user = None
    dialog.append(same_user_utterances)

    for dialog_line in dialog_reader:

        if first_turn:
            last_user = dialog_line[1]
            first_turn = False

        if last_user != dialog_line[1]:
            # user has changed
            same_user_utterances = []
            dialog.append(same_user_utterances)

        same_user_utterances.append(dialog_line[3])

        last_user = dialog_line[1]

    dialog.append([dialog_end_symbol])

    return dialog
```

```
def
get_random_utterances_from_corpus(candidate_dialog_paths,rng,utterances_num=9,min
_turn=3,max_turn=20):
    """
    Sample multiple random utterances from the whole corpus.
    :param candidate_dialog_paths:
    :param rng:
    :param utterances_num: number of utterances to generate
    :param min_turn: minimal index of turn that the utterance is selected from
    :return:
    """
    utterances = []
    dialogs_num = len(candidate_dialog_paths)

    for i in xrange(0,utterances_num):
        # sample random dialog
        dialog_path = candidate_dialog_paths[rng.randint(0,dialogs_num-1)]
        # load the dialog
        dialog = translate_dialog_to_lists(dialog_path)

        # we do not count the last  _dialog_end__ urn
        dialog_len = len(dialog) - 1
        if(dialog_len<min_turn):
            print("Dialog {} was shorter than the minimum required lenght
{}".format(dialog_path,dialog_len))
            exit()
        # sample utterance, exclude the last round that is always "dialog end"
        max_ix = min(max_turn, dialog_len) - 1

        # this code deals with corner cases when dialogs are too short
        if min_turn -1 == max_ix:
            turn_index = max_ix
        else:
            turn_index = rng.randint(min_turn,max_ix)

        utterance = singe_user_utterances_to_string(dialog[turn_index])
        utterances.append(utterance)
    return utterances

def singe_user_utterances_to_string(utterances_list):
    """
    Concatenates multiple user's utterances into a single string.
    :param utterances_list:
    :return:
    """
    return " ".join(map(lambda x: x+" "+ end_of_utterance_symbol,
utterances_list))

def dialog_turns_to_string(dialog):
    """
    Translates the whole dialog (list of lists) to a string
```

```
    :param dialog:
    :return:
    """
    # join utterances
    turns_as_strings = map(singe_user_utterances_to_string,dialog)
    # join turns
    return "".join(map(lambda x : x + " " + end_of_turn_symbol + " ",
turns_as_strings))

def
create_random_context(dialog,rng,minimum_context_length=2,max_context_length=20):
    """
    Samples random context from a dialog. Contexts are uniformly sampled from the
whole dialog.
    :param dialog:
    :param rng:
    :return: context, index of next utterance that follows the context
    """
    # sample dialog context
    #context_turns = rng.randint(minimum_context_length,len(dialog)-1)
    max_len = min(max_context_length, len(dialog)) - 2
    if max_len <= minimum_context_length:
        context_turns = max_len
    else:
        context_turns = rng.randint(minimum_context_length,max_len)

    # create string
    return dialog_turns_to_string(dialog[:context_turns]),context_turns



def create_single_dialog_train_example(context_dialog_path,
candidate_dialog_paths, rng, positive_probability,

minimum_context_length=2,max_context_length=20):
    """
    Creates a single example for training set.
    :param context_dialog_path:
    :param candidate_dialog_paths:
    :param rng:
    :param positive_probability:
    :return:
    """

    dialog = translate_dialog_to_lists(context_dialog_path)

    context_str, next_utterance_ix = create_random_context(dialog, rng,

minimum_context_length=minimum_context_length,

max_context_length=max_context_length)
```

```
    if positive_probability > rng.random():
        # use the next utterance as positive example
        response = singe_user_utterances_to_string(dialog[next_utterance_ix])
        label = 1.0
    else:
        response =
get_random_utterances_from_corpus(candidate_dialog_paths,rng,1,

min_turn=minimum_context_length+1,

max_turn=max_context_length)[0]
        label = 0.0
    return context_str, response, label


def create_single_dialog_test_example(context_dialog_path,
candidate_dialog_paths, rng, distractors_num, max_context_length):
    """
    Creates a single example for testing or validation. Each line contains a
context, one positive example and N negative examples.
    :param context_dialog_path:
    :param candidate_dialog_paths:
    :param rng:
    :param distractors_num:
    :return: triple (context, positive response, [negative responses])
    """

    dialog = translate_dialog_to_lists(context_dialog_path)

    context_str, next_utterance_ix = create_random_context(dialog, rng,
max_context_length=max_context_length)

    # use the next utterance as positive example
    positive_response =
singe_user_utterances_to_string(dialog[next_utterance_ix])

    negative_responses =
get_random_utterances_from_corpus(candidate_dialog_paths,rng,distractors_num)
    return context_str, positive_response, negative_responses


def create_examples_train(candidate_dialog_paths, rng, positive_probability=0.5,
max_context_length=20):
    """
    Creates single training example.
    :param candidate_dialog_paths:
    :param rng:
    :param positive_probability: probability of selecting positive training
example
    :return:
```

```
    """
    i = 0
    examples = []
    for context_dialog in candidate_dialog_paths:
        if i % 1000 == 0:
            print(str(i))
        dialog_path = candidate_dialog_paths[i]
        examples.append(create_single_dialog_train_example(dialog_path,
candidate_dialog_paths, rng, positive_probability,

max_context_length=max_context_length))
        i+=1
    #return map(lambda dialog_path :
create_single_dialog_train_example(dialog_path, candidate_dialog_paths, rng,
positive_probability), candidate_dialog_paths)

def create_examples(candidate_dialog_paths, examples_num, creator_function):
    """
    Creates a list of training examples from a list of dialogs and function that
transforms a dialog to an example.
    :param candidate_dialog_paths:
    :param creator_function:
    :return:
    """
    i = 0
    examples = []
    unique_dialogs_num = len(candidate_dialog_paths)

    while i < examples_num:
        context_dialog = candidate_dialog_paths[i % unique_dialogs_num]
        # counter for tracking progress
        if i % 1000 == 0:
            print(str(i))
        i+=1

        examples.append(creator_function(context_dialog, candidate_dialog_paths))

    return examples

def convert_csv_with_dialog_paths(csv_file):
    """
    Converts CSV file with comma separated paths to filesystem paths.
    :param csv_file:
    :return:
    """
    def convert_line_to_path(line):
        file, dir = map(lambda x : x.strip(), line.split(","))
        return os.path.join(dir, file)

    return map(convert_line_to_path, csv_file)
```

```
def prepare_data_maybe_download(directory):
  """
  Download and unpack dialogs if necessary.
  """
  filename = 'ubuntu_dialogs.tgz'
  url = 'http://cs.mcgill.ca/~jpineau/datasets/ubuntu-corpus-
1.0/ubuntu_dialogs.tgz'
  dialogs_path = os.path.join(directory, 'dialogs')

  # test it there are some dialogs in the path
  if not os.path.exists(os.path.join(directory,"10","1.tst")):
    # dialogs are missing
    archive_path = os.path.join(directory,filename)
    if not os.path.exists(archive_path):
        # archive missing, download it
        print("Downloading %s to %s" % (url, archive_path))
        filepath, _ = urllib.request.urlretrieve(url, archive_path)
        print("Successfully downloaded " + filepath)

    # unpack data
    if not os.path.exists(dialogs_path):
          print("Unpacking dialogs ...")
          with tarfile.open(archive_path) as tar:
                tar.extractall(path=directory)
          print("Archive unpacked.")

    return

if __name__ == '__main__':


    def create_eval_dataset(args, file_list_csv):
        rng = random.Random(args.seed)
        # training dataset
        f = open(os.path.join("meta", file_list_csv), 'r')
        dialog_paths = map(lambda path: os.path.join(args.data_root, "dialogs",
path), convert_csv_with_dialog_paths(f))

        data_set = create_examples(dialog_paths,
                                   len(dialog_paths),
                                   lambda context_dialog, candidates :
create_single_dialog_test_example(context_dialog, candidates, rng,
                                                                    args.n,
args.max_context_length))
        # output the dataset
        w = unicodecsv.writer(open(args.output, 'w'), encoding='utf-8')
        # header
        header = ["Context", "Ground Truth Utterance"]
        header.extend(map(lambda x: "Distractor_{}".format(x), xrange(args.n)))
        w.writerow(header)
```

```
        stemmer = SnowballStemmer("english")
        lemmatizer = WordNetLemmatizer()

        for row in data_set:
            translated_row = [row[0], row[1]]
            translated_row.extend(row[2])

            if args.tokenize:
                translated_row = map(nltk.word_tokenize, translated_row)
                if args.stem:
                    translated_row = map(lambda sub: map(stemmer.stem, sub),
translated_row)
                if args.lemmatize:
                    translated_row = map(lambda sub: map(lambda tok:
lemmatizer.lemmatize(tok, pos='v'), sub), translated_row)

                translated_row = map(lambda x: " ".join(x), translated_row)

            w.writerow(translated_row)
        print("Dataset stored in: {}".format(args.output))


    def train_cmd(args):

        rng = random.Random(args.seed)
        # training dataset

        f = open(os.path.join("meta", "trainfiles.csv"), 'r')
        dialog_paths = map(lambda path: os.path.join(args.data_root, "dialogs",
path), convert_csv_with_dialog_paths(f))

        train_set = create_examples(dialog_paths,
                                    args.examples,
                                    lambda context_dialog, candidates :

create_single_dialog_train_example(context_dialog, candidates, rng,
                                                                    args.p,
max_context_length=args.max_context_length))

        stemmer = SnowballStemmer("english")
        lemmatizer = WordNetLemmatizer()

        # output the dataset
        w = unicodecsv.writer(open(args.output, 'w'), encoding='utf-8')
        # header
        w.writerow(["Context", "Utterance", "Label"])
        for row in train_set:
            translated_row = row

            if args.tokenize:
                translated_row = [nltk.word_tokenize(row[i]) for i in [0,1]]
```

```
                if args.stem:
                    translated_row = map(lambda sub: map(stemmer.stem, sub),
translated_row)
                if args.lemmatize:
                    translated_row = map(lambda sub: map(lambda tok:
lemmatizer.lemmatize(tok, pos='v'), sub), translated_row)

                translated_row = map(lambda x: " ".join(x), translated_row)
                translated_row.append(int(float(row[2])))

            w.writerow(translated_row)
        print("Train dataset stored in: {}".format(args.output))
```

Заключение

В ходе курсовой работы я реализовал диалоговый чат бот на языке Python с использованием двух подходов. Проблемы, с которыми я столкнулся, я уже описал. Было проведено тестирование, результаты удовлетворительны, но тест Тьюринга данный бот явно не пройдет. В ходе работы, я вспомнил, что из себя представляет BF, Telegram API и почему я им больше не пользуюсь. Снова потрогал алгоритм TF-IDF и базы данных и занялся дата майнингом