5.3. Руководство к практическому занятию 3.

Содержание

Цель работы. Научиться основам разработки чат-ботов, ведущих диалог с пользователем на разные темы, но имеющий определенную задачу – ненавязчиво информировать пользователя о характеристиках/достоинствах какого-то продукта/товара и по возможности предложить пользователю его покупку.

---

1. Основные задачи на базе методов обработки естественного языка (Nature Language Processing - NLP)

1.1. Обработка ввода

После ввода сообщения пользователя необходимо проделать следующие действия:

1. Очистка ввода (удаление лишних пробелов, случайных символов и т.п.), работа с опечатками (использование словарей, учет расстояния Левенштейна при поиске) и т.п. Под расстоянием Ливенштейна понимается минимальное количество операций удаления, вставки и замены символа, необходимое для преобразования одной строки (введенной с опечатками) в другую (эталонная, из базы данных/репозитория строк). Этот этап в англоязычной литературе часто называется *spell correction*.
2. Лемматизация - это процесс преобразования слова в его базовую форму (избавление от спряжений, склонений, множественных форм и т.д.). Упрощенная форма лемматизации – стемминг. Это когда в слове удаляются последние несколько символов, связанные с окончанием и/или суффиксом, оставляя корень слова. Однако такое упрощение нужно использовать с осторожностью, т.к. это может привести к некоторым ошибкам, например, «Caring» -> Лемматизация -> «Care» (заботиться), «Caring» -> Стемминг -> «Car»(машина). Элементы лемматизации есть в открытой Библиотеке Natasha.
3. Классификация намерений (intent classification, intent extraction). Этот этап позволяет выявить в фразах пользователя его намерения, желания, вопросы и в соответствии с этими ожиданиями пользователя вести диалог. В простейшем случае составляется словарь, который сопоставляет некоторые типовые фразы, словосочетания, слова множеству возможных намерений.
4. Извлечение сущностей. Для этого может быть использована библиотека Natasha. Библиотека Natasha решает базовые задачи обработки естественного русского языка: сегментация на токены и предложения, морфологический и синтаксический анализ, лемматизация, извлечение именованных сущностей. Для новостных статей качество на всех задачах сравнимо или превосходит существующие решения. Библиотека поддерживает Python 3.5+ и PyPy3, не требует GPU, зависит только от NumPy.
5. Сентимент анализ или анализ тональности. Как минимум: положительный, нейтральный или отрицательный тон сообщения. Вводится шкала, например от -1 до +1. Более подробный анализ позволяет выявить детали тональности (взволновано, шутливо, нейтрально, угрожающе, и т.д.). В простейшей реализации используются словари с заранее прописанными коэффициентами тональности для слов. Например, «люблю: +1; ненавижу: -1, пойду: 0». Скачать датасет тонального словаря русского языка можно тут (примерно 28 тыс.слов): https://github.com/dkulagin/kartaslov/tree/master/dataset/emo_dict

6. Классификация по темам. Бывает бинарная классификация и мультиклассовая классификация. Бинарная позволяет определить – соответствует ли фраза (текст) нужной тематике или нет. Мультиклассовая определяет для каждого класса вероятность (близость) фразы(текста) к соответствующему классу, описывающему какую-то тему. Простейшая реализация – на основе словаря терминов, словосочетаний, слов привязанных к некоторым темам.
7. Распознавание речи. Используются библиотеки SpeechRecognition, gTTS, PyAudio.

1.2. Генерация ответа

1.2.1. Алгоритмы выбора заготовленных фраз и реплик.

1.2.2. Алгоритмы действия по сценарию (случайный или по контексту переход к рекламе или нужной тематике).

1.2.3. Использование заранее подготовленных правил.

1.2.4. Автоматическая генерация реплики по контексту.

1.2.5. Синтез речи.

2.Расширение интеллектуальных возможностей чат-бота

2.1. Использование машинного обучения (ML) для анализа намерений

Чтобы создать и обучить нейронную сеть для анализа намерений пользователя в ходе диалога нужно осуществить следующие действия:

1. Провести векторизацию фразы. Векторизация текста – это способ перевода текстов в числовые тензоры, или векторы чисел. Делается для того, чтобы подать эти векторы на вход нейронной сети – классификатора. Можно использовать библиотеку Sklearn.
2. Создать классификатор на основе нейронной сети.
3. Обучить нейронную сеть.
4. Подать на нейронную сеть введенную фразу (предварительно тоже векторизованную).
5. Получить ответ нейронной сети – классификация намерения.

Пример реализации этих шагов на основе фреймворка Sklearn.:

```
import random
import nltk
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer

#создаем векторайзер

vectorizer = TfidfVectorizer(analyzer='char', ngram_range=(3, 3))

#преобразуем исходный X_text в вектор X
X = vectorizer.fit_transform(X_text)

#создаем линейный классификатор
clf = LinearSVC()

#обучаем нейронную сеть

clf.fit(X, y)

. . . . . .

#далее используем нейронку для предсказания (прогноза) намерения

intent = clf.predict(vectorizer.transform([replica]))[0]
```

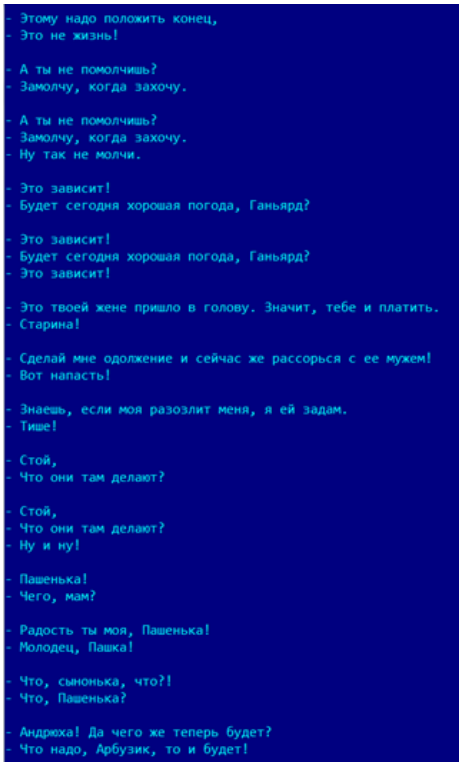
-

2.2. Использование стандартных диалогов из книг/интернета для поиска типовых ответов

Можно настроить чат-бот так, чтобы он искал в текстовом датасете типовые вопросы и давал ответы. Для этого должен быть большой датасет, состоящий из пар строк: вопрос-ответ. Чат-бот ищет наиболее подходящий (близкий по смыслу) вопрос и выдает заготовленный ответ.

В сети на github можно найти такой датасет – dialogues.txt. Правее приведен фрагмент из датасета диалогов.

Исходный датасет достаточно «сырой», его нужно подготовить: убрать длинные диалоги (оставить только первые две строки: вопрос – ответ; почистить повторы вопросов. Вот примерная реализация работы с таким датасетом: текстовый файл – dialogues.txt (133M).




```
with open('dialogues.txt') as f:
    content = f.read()
#разбиваем на пары по признаку двойного перевода строки

dialogues_str = content.split('\n\n')

dialogues = [dialogue_str.split('\n')[:2] for dialogue_str in dialogues_str]


dialogues_filtered = []
questions = set()


#убираем все после второй строки

for dialogue in dialogues:
    if len(dialogue) != 2:
        continue

    question, answer = dialogue
    question = clear_phrase(question[2:])
    answer = answer[2:]

    if question != '' and question not in questions:
        questions.add(question)
        dialogues_filtered.append([question, answer])
```

```
#формируем структурированные диалоги см.формат ниже
dialogues_structured = {}  #  {'word': [['...word...', 'answer'], ...], ...}

for question, answer in dialogues_filtered:
    words = set(question.split(' '))
    for word in words:
        if word not in dialogues_structured:
            dialogues_structured[word] = []
        dialogues_structured[word].append([question, answer])

dialogues_structured_cut = {}
for word, pairs in dialogues_structured.items():
    pairs.sort(key=lambda pair: len(pair[0]))
    dialogues_structured_cut[word] = pairs[:1000]


# replica -> word1, word2, word3, ... -> dialogues_structured[word1] + dialogues_structured[word2] + ... -> mini_dataset


#так выглядит функция генерации ответа с использованием датасета диалогов
def generate_answer(replica):
    replica = clear_phrase(replica)
    words = set(replica.split(' '))
    mini_dataset = []
    for word in words:
        if word in dialogues_structured_cut:
            mini_dataset += dialogues_structured_cut[word]

    # TODO убрать повторы из mini_dataset!

    answers = []  # [[distance_weighted, question, answer]]

    for question, answer in mini_dataset:
        if abs(len(replica) - len(question)) / len(question) < 0.2:
            distance = nltk.edit_distance(replica, question)
            distance_weighted = distance / len(question)
            if distance_weighted < 0.2:
                answers.append([distance_weighted, question, answer])

    if answers:
        return min(answers, key=lambda three: three[0])[2]
```

2.3. Настройка чат-бота в мессенджере «Телеграмм»

```
# https://github.com/python-telegram-bot/python-telegram-bot

get_ipython().system(' pip install python-telegram-bot --upgrade')


from telegram import Update
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters, CallbackContext


def start(update: Update, context: CallbackContext) -> None:
    """Send a message when the command /start is issued."""
    update.message.reply_text('Hi!')


def help_command(update: Update, context: CallbackContext) -> None:
    """Send a message when the command /help is issued."""
    update.message.reply_text('Help!')
```

```
def run_bot(update: Update, context: CallbackContext) -> None:
    replica = update.message.text
    answer = bot(replica)
    update.message.reply_text(answer)

    print(stats)
    print(replica)
    print(answer)
    print()


def main():
    """Start the bot."""

    updater = Updater("1099502691:AAEdBLT5DY9by6z_Qi8UpyOpaxZfJBVsw9c")

    dispatcher = updater.dispatcher
    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(CommandHandler("help", help_command))
    dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, run_bot))

    # Start the Bot
    updater.start_polling()
    updater.idle()


main()
```

3. Примеры чат-ботов различной сложности

Рассмотрим примеры кода для различных частей интеллектуального чат-бота, постепенно повышая сложность.

3.1. Пример бота 1: уровень «ниже минимума»

```
import random
import nltk

BOT_CONFIG = { #Здесь определяем настройки бота и первое - словарь намерений
    'intents': { #начинается словарь намерений

        'hello': { #намерение 1: приветствие (hello)

#следующая строка – возможные входные фразы
          'examples': ['Привет', 'Добрый день', 'Как дела', 'Привет, бот'],

#а здесь указываются возможные ответы на это намерение
          'responses': ['Привет, человек!', 'Здравствуйте !)', 'Доброго времени суток']
        },
        'bye': { #следующее намерение 2 - прощание
          'examples': ['Пока', 'До свидания', 'До свидания', 'До скорой встречи'],
          'responses': ['Еще увидимся', 'Если что, я всегда тут']
        },
        'name': {  #следующее намерение 3 - знакомство
          'examples': ['Как тебя зовут?', 'Скажи свое имя', 'Представься'],
          'responses': ['Меня зовут Саша']
        },
    },
#тут далее вы добавляете свои намерения и расширяете существующие.
    'failure_phrases': [ #а это стандартные ответы, если бот не понял намерения
        'Непонятно. Перефразируйте, пожалуйста.',
```

```
        'Я еще только учусь. Спросите что-нибудь другое',
        'Слишком сложный вопрос для меня.',
    ]
}
def clear_phrase(phrase): #функция простейшей очистки фраз от «мусора»
    phrase = phrase.lower()
    alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя- '
    result = ''.join(symbol for symbol in phrase if symbol in alphabet)
#     result = ''
#     for symbol in phrase:
#         if symbol in alphabet:
#             result += symbol
# TODO: Вам необходимо существенно доработать эту функцию !

    return result
def classify_intent(replica): #определение намерения
    # TODO use ML! – добавить машинное обучение !!!
    replica = clear_phrase(replica)
#ищем по словарю намерений !
    for intent, intent_data in BOT_CONFIG['intents'].items():
        for example in intent_data['examples']:
            example = clear_phrase(example)
#используем расстояние Левенштейна
            distance = nltk.edit_distance(replica, example)
            if distance / len(example) < 0.4:
                return intent
#функция выдает случайные ответ по известному намерению из словаря намерений
def get_answer_by_intent(intent):
    if intent in BOT_CONFIG['intents']:
        responses = BOT_CONFIG['intents'][intent]['responses']
        return random.choice(responses)

def generate_answer(replica):
# TODO : Это заглушка генеративной модели, но для более высокой оценки – нужно реализовать!

    return

def get_failure_phrase(): #выдать случайную фразу на неизвестное намерение
    failure_phrases = BOT_CONFIG['failure_phrases']
    return random.choice(failure_phrases)


#ОСНОВНАЯ ФУНКЦИЯ БОТА – по фразе (replica) выдать ответ
def bot(replica):
    # NLU – Natural Language Understanding (понимание естественного языка)
    intent = classify_intent(replica)#проверяем намерения

    # Answer generation

    # выбор заготовленной реплики
    if intent:
        answer = get_answer_by_intent(intent)
        if answer:
            return answer

    # вызов генеративной модели
    answer = generate_answer(replica)
    if answer:
        return answer

    # если не понятно намерение – выдаем стандартную фразу
    return get_failure_phrase()
```

```
#проверяем работу бота на некоторой фразе
bot('добрый день')
```

3.2. Пример бота 2: уровень «минимум»

В этом боте расширяются некоторые возможности по сравнению с ботом 1. Но все еще нет обучающей модели.

```
import random
import nltk

# В структурах intents ключ theme_gen определяет тему которую инициирует данное намерение
# ключ theme_app определяет к каким темам применим данное намерение (значение * применяет намерение к любой теме сразу)
# если данных ключей нет, то намерение применимо к ситуации когда тема не определена
# история тем накапливается в списке hist_theme
# поиск намерения по теме производится по темам в истории тем начиная от последней до самой ранней, если не обнаружено, то производится поиск без темы

BOT_CONFIG = {
    'intents': {
        'hello': {
            'examples': ['Привет', 'Добрый день', 'Шалом', 'Привет, бот'],
            'responses': ['Привет, человек!', 'И вам здравствуйте :)', 'Доброго времени суток']
        },
        'bye': {
            'examples': ['Пока', 'До свидания', 'До свидания', 'До скорой встречи'],
            'responses': ['Еще увидимся', 'Если что, я всегда тут']
        },
        'name': {
            'examples': ['Как тебя зовут?', 'Скажи свое имя', 'Представься'],
            'responses': ['Меня зовут Саша']
        },
        'want_eat': {
            'examples': ['Хочу есть', 'Хочу кушать', 'ням-ням'],
            'responses': ['Вы веган?'],
            'theme_gen': 'eating_q_wegan',
            'theme_app': ['eating', '*']
        },
        'yes': {
            'examples': ['да'],
            'responses': ['капусты или морковки?'],
            'theme_gen': 'eating_q_meal',
            'theme_app': ['eating_q_wegan']
        },
        'no': {
            'examples': ['нет'],
            'responses': ['мясо или творог?'],
            'theme_gen': 'eating_q_meal',
            'theme_app': ['eating_q_wegan']
        },
    },

    'failure_phrases': [
        'Непонятно. Перефразируйте, пожалуйста.',
        'Я еще только учусь. Спросите что-нибудь другое',
        'Слишком сложный вопрос для меня.',
    ]
}

HIST_THEME_LEN = 10
hist_theme = []

def clear_phrase(phrase):
    phrase = phrase.lower()
```

```
    alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя- '
    result = ''.join(symbol for symbol in phrase if symbol in alphabet)

    #     result = ''
    #     for symbol in phrase:
    #         if symbol in alphabet:
    #             result += symbol

    return result

def classify_intent_by_theme(replica, theme=None):
    # TODO use ML!

    replica = clear_phrase(replica)

    for intent, intent_data in BOT_CONFIG['intents'].items():

        theme_app = None
        if 'theme_app' in intent_data:
            theme_app = intent_data['theme_app']

        if (theme_app is not None and (theme in theme_app or '*' in theme_app)) or ( theme is None and theme_app is None):
            for example in intent_data['examples']:
                example = clear_phrase(example)

                distance = nltk.edit_distance(replica, example)
                if distance / len(example) < 0.4:
                    return intent

def classify_intent(replica):
    global hist_theme
    lev = 0
    intent = None
    # Перебор истории тем
    for theme in hist_theme:
        intent = classify_intent_by_theme(replica, theme)
        if intent is not None:
            break
        lev += 1

    if intent is None:  # Если по темам не обнаружено намерений, то ищем без темы
        lev = 0  # Чтобы не очистить историю тем (можно и как вариант очищать, чтобы при непонятных ситуациях забывать историю)
        intent = classify_intent_by_theme(replica)
    else:
        if lev > 0:
            hist_theme = hist_theme[lev:]  # Перескок на более старую тему, если определеили её

    if intent is not None:
        if 'theme_gen' in BOT_CONFIG['intents'][intent]:  # Если намерение генерирует новую тему
            if BOT_CONFIG['intents'][intent]['theme_gen'] not in hist_theme:  # И её нет ещё в истории
                hist_theme.insert(0, BOT_CONFIG['intents'][intent]['theme_gen'])  # То добавляем в историю тему
                if (len(hist_theme) > HIST_THEME_LEN):
                    hist_theme.pop()  # Ограничение длины истории тем
    return intent

def get_answer_by_intent(intent):
    if intent in BOT_CONFIG['intents']:
        responses = BOT_CONFIG['intents'][intent]['responses']
        return random.choice(responses)

def generate_answer(replica):
    # TODO
    return

def get_failure_phrase():
```

```
    failure_phrases = BOT_CONFIG['failure_phrases']
    return random.choice(failure_phrases)

def bot(replica):
    # NLU - Natural-Language Understanding, понимание естественного языка
    intent = classify_intent(replica)
    # Генерация ответа
    # выбор заготовленной реплики
    if intent:
        answer = get_answer_by_intent(intent)
        if answer:
            return answer

    # вызов генеративной модели
    answer = generate_answer(replica)
    if answer:
        return answer

    # берем заглушку
    return get_failure_phrase()

#bot('добрый вечер')

#создаем основной цикл работы бота

question = None
hist_theme = []

while (True):
    question = input('> ')
    if question != '':
        answer = bot(question)
        if answer is not None:
            print('< ' + answer)
            # print(hist_theme)
    else:
        break
```

3.3. Пример бота 3: уровень «средний»

В данном примере реализован бот с обученной нейронной сетью и поиском ответов по датасету с фрагментами текстов диалогов из книг/интернет.

```
#!/usr/bin/env python
# coding: utf-8

import random
import nltk
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer


BOT_CONFIG = {
    'intents': {

        'hello': {
            'examples': ['Привет', 'Добрый день', 'Шалом', 'Привет, бот'],
            'responses': ['Привет, человек!', 'И вам здравствуйте :)', 'Доброго времени суток']
        },
        'bye': {
            'examples': ['Пока', 'До свидания', 'До свидания', 'До скорой встречи'],
            'responses': ['Еще увидимся', 'Если что, я всегда тут']
        },
```

```
        'name': {
            'examples': ['Как тебя зовут?', 'Скажи свое имя', 'Представься'],
            'responses': ['Меня зовут Саша']
        },
        'want_eat': {
            'examples': ['Хочу есть', 'Хочу кушать', 'ням-ням'],
            'responses': ['Вы веган?'],
            'theme_gen': 'eating_q_wegan',
            'theme_app': ['eating', '*']
        },
        'yes': {
            'examples': ['да'],
            'responses': ['капусты или морковки?'],
            'theme_gen': 'eating_q_meal',
            'theme_app': ['eating_q_wegan']
        },
        'no': {
            'examples': ['нет'],
            'responses': ['мясо или творог?'],
            'theme_gen': 'eating_q_meal',
            'theme_app': ['eating_q_wegan']
        },
    },

    'failure_phrases': [
        'Непонятно. Перефразируйте, пожалуйста.',
        'Я еще только учусь. Спросите что-нибудь другое',
        'Слишком сложный вопрос для меня.',
    ]
}


X_text = []  # ['Хэй', 'хаюхай', 'Хаюшки', ...]
y = []  # ['hello', 'hello', 'hello', ...]

for intent, intent_data in BOT_CONFIG['intents'].items():
    for example in intent_data['examples']:
        X_text.append(example)
        y.append(intent)

vectorizer = TfidfVectorizer(analyzer='char', ngram_range=(3, 3))
X = vectorizer.fit_transform(X_text)
clf = LinearSVC()
clf.fit(X, y)

def clear_phrase(phrase):
    phrase = phrase.lower()

    alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя- '
    result = ''.join(symbol for symbol in phrase if symbol in alphabet)

    return result.strip()

def classify_intent(replica):
    replica = clear_phrase(replica)

    intent = clf.predict(vectorizer.transform([replica]))[0]

    for example in BOT_CONFIG['intents'][intent]['examples']:
        example = clear_phrase(example)
        distance = nltk.edit_distance(replica, example)
        if example and distance / len(example) <= 0.5:
            return intent

def get_answer_by_intent(intent):
```

```
    if intent in BOT_CONFIG['intents']:
        responses = BOT_CONFIG['intents'][intent]['responses']
        if responses:
            return random.choice(responses)

with open('dialogues.txt') as f:
    content = f.read()

dialogues_str = content.split('\n\n')
dialogues = [dialogue_str.split('\n')[:2] for dialogue_str in dialogues_str]

dialogues_filtered = []
questions = set()

for dialogue in dialogues:
    if len(dialogue) != 2:
        continue

    question, answer = dialogue
    question = clear_phrase(question[2:])
    answer = answer[2:]

    if question != '' and question not in questions:
        questions.add(question)
        dialogues_filtered.append([question, answer])

dialogues_structured = {}  #  {'word': [['...word...', 'answer'], ...], ...}

for question, answer in dialogues_filtered:
    words = set(question.split(' '))
    for word in words:
        if word not in dialogues_structured:
            dialogues_structured[word] = []
        dialogues_structured[word].append([question, answer])

dialogues_structured_cut = {}
for word, pairs in dialogues_structured.items():
    pairs.sort(key=lambda pair: len(pair[0]))
    dialogues_structured_cut[word] = pairs[:1000]

# replica -> word1, word2, word3, ... -> dialogues_structured[word1] + dialogues_structured[word2] + ... -> mini_dataset

def generate_answer(replica):
    replica = clear_phrase(replica)
    words = set(replica.split(' '))
    mini_dataset = []
    for word in words:
        if word in dialogues_structured_cut:
            mini_dataset += dialogues_structured_cut[word]

    # TODO убрать повторы из mini_dataset

    answers = []  # [[distance_weighted, question, answer]]

    for question, answer in mini_dataset:
        if abs(len(replica) - len(question)) / len(question) < 0.2:
            distance = nltk.edit_distance(replica, question)
            distance_weighted = distance / len(question)
            if distance_weighted < 0.2:
                answers.append([distance_weighted, question, answer])

    if answers:
        return min(answers, key=lambda three: three[0])[2]

def get_failure_phrase():
```

```
    failure_phrases = BOT_CONFIG['failure_phrases']
    return random.choice(failure_phrases)

stats = {'intent': 0, 'generate': 0, 'failure': 0}

def bot(replica):
    # NLU
    intent = classify_intent(replica)

    # Answer generation

    # выбор заготовленной реплики
    if intent:
        answer = get_answer_by_intent(intent)
        if answer:
            stats['intent'] += 1
            return answer
    # вызов генеративной модели
    answer = generate_answer(replica)
    if answer:
        stats['generate'] += 1
        return answer

    # берем заглушку
    stats['failure'] += 1
    return get_failure_phrase()

bot('кто ищет тот всегда найдет!')
```

3.4. Пример бота 4: уровень «выше среднего»

В данном примере реализован бот с обученной нейронной сетью и поиском ответов по датасету с фрагментами текстов диалогов из книг/интернет, а также с подключением к мессенджеру «Телеграмм».

```
#!/usr/bin/env python
# coding: utf-8
import random
import nltk
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer

BOT_CONFIG = {
    'intents': {

        'hello': {
            'examples': ['Привет', 'Добрый день', 'Шалом', 'Привет, бот'],
            'responses': ['Привет, человек!', 'И вам здравствуйте :)', 'Доброго времени суток']
        },
        'bye': {
            'examples': ['Пока', 'До свидания', 'До свидания', 'До скорой встречи'],
            'responses': ['Еще увидимся', 'Если что, я всегда тут']
        },
        'name': {
            'examples': ['Как тебя зовут?', 'Скажи свое имя', 'Представься'],
            'responses': ['Меня зовут Саша']
        },
        'want_eat': {
            'examples': ['Хочу есть', 'Хочу кушать', 'ням-ням'],
            'responses': ['Вы веган?'],
            'theme_gen': 'eating_q_wegan',
            'theme_app': ['eating', '*']
        },
        'yes': {
```

```
            'examples': ['да'],
            'responses': ['капусты или морковки?'],
            'theme_gen': 'eating_q_meal',
            'theme_app': ['eating_q_wegan']
        },
        'no': {
            'examples': ['нет'],
            'responses': ['мясо или творог?'],
            'theme_gen': 'eating_q_meal',
            'theme_app': ['eating_q_wegan']
        },
    },

    'failure_phrases': [
        'Непонятно. Перефразируйте, пожалуйста.',
        'Я еще только учусь. Спросите что-нибудь другое',
        'Слишком сложный вопрос для меня.',
    ]
}

X_text = []  # ['Хэй', 'хаюхай', 'Хаюшки', ...]
y = []  # ['hello', 'hello', 'hello', ...]

for intent, intent_data in BOT_CONFIG['intents'].items():
    for example in intent_data['examples']:
        X_text.append(example)
        y.append(intent)

vectorizer = TfidfVectorizer(analyzer='char', ngram_range=(3, 3))
X = vectorizer.fit_transform(X_text)
clf = LinearSVC()
clf.fit(X, y)

def clear_phrase(phrase):
    phrase = phrase.lower()

    alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя- '
    result = ''.join(symbol for symbol in phrase if symbol in alphabet)

    return result.strip()

def classify_intent(replica):
    replica = clear_phrase(replica)

    intent = clf.predict(vectorizer.transform([replica]))[0]

    for example in BOT_CONFIG['intents'][intent]['examples']:
        example = clear_phrase(example)
        distance = nltk.edit_distance(replica, example)
        if example and distance / len(example) <= 0.5:
            return intent

def get_answer_by_intent(intent):
    if intent in BOT_CONFIG['intents']:
        responses = BOT_CONFIG['intents'][intent]['responses']
        if responses:
            return random.choice(responses)

with open('dialogues.txt') as f:
    content = f.read()

dialogues_str = content.split('\n\n')
dialogues = [dialogue_str.split('\n')[:2] for dialogue_str in dialogues_str]

dialogues_filtered = []
```

```
questions = set()

for dialogue in dialogues:
    if len(dialogue) != 2:
        continue

    question, answer = dialogue
    question = clear_phrase(question[2:])
    answer = answer[2:]

    if question != '' and question not in questions:
        questions.add(question)
        dialogues_filtered.append([question, answer])

dialogues_structured = {}  #  {'word': [['...word...', 'answer'], ...], ...}

for question, answer in dialogues_filtered:
    words = set(question.split(' '))
    for word in words:
        if word not in dialogues_structured:
            dialogues_structured[word] = []
        dialogues_structured[word].append([question, answer])

dialogues_structured_cut = {}
for word, pairs in dialogues_structured.items():
    pairs.sort(key=lambda pair: len(pair[0]))
    dialogues_structured_cut[word] = pairs[:1000]

# replica -> word1, word2, word3, ... -> dialogues_structured[word1] + dialogues_structured[word2] + ... -> mini_dataset

def generate_answer(replica):
    replica = clear_phrase(replica)
    words = set(replica.split(' '))
    mini_dataset = []
    for word in words:
        if word in dialogues_structured_cut:
            mini_dataset += dialogues_structured_cut[word]

    # TODO убрать повторы из mini_dataset

    answers = []  # [[distance_weighted, question, answer]]

    for question, answer in mini_dataset:
        if abs(len(replica) - len(question)) / len(question) < 0.2:
            distance = nltk.edit_distance(replica, question)
            distance_weighted = distance / len(question)
            if distance_weighted < 0.2:
                answers.append([distance_weighted, question, answer])

    if answers:
        return min(answers, key=lambda three: three[0])[2]

def get_failure_phrase():
    failure_phrases = BOT_CONFIG['failure_phrases']
    return random.choice(failure_phrases)

stats = {'intent': 0, 'generate': 0, 'failure': 0}

def bot(replica):
    # NLU
    intent = classify_intent(replica)

    # Answer generation

    # выбор заготовленной реплики
```

```
    if intent:
        answer = get_answer_by_intent(intent)
        if answer:
            stats['intent'] += 1
            return answer

    # вызов генеративной модели
    answer = generate_answer(replica)
    if answer:
        stats['generate'] += 1
        return answer

    # берем заглушку
    stats['failure'] += 1
    return get_failure_phrase()

bot('Сколько времени?')

############### ТЕЛЕГРАММ ############################

# https://github.com/python-telegram-bot/python-telegram-bot

get_ipython().system(' pip install python-telegram-bot --upgrade')

from telegram import Update
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters, CallbackContext

def start(update: Update, context: CallbackContext) -> None:
    """Send a message when the command /start is issued."""
    update.message.reply_text('Hi!')

def help_command(update: Update, context: CallbackContext) -> None:
    """Send a message when the command /help is issued."""
    update.message.reply_text('Help!')

def run_bot(update: Update, context: CallbackContext) -> None:
    replica = update.message.text
    answer = bot(replica)
    update.message.reply_text(answer)

    print(stats); print(replica); print(answer); print()

def main():
    """Start the bot."""
    updater = Updater("1099502691:AAEdBLT5DY9by6z_Qi8UpyOpaxZfJBVsw9c")
    dispatcher = updater.dispatcher
    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(CommandHandler("help", help_command))
    dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, run_bot))
    # Start the Bot
    updater.start_polling()
    updater.idle()

main()
```

Последнее изменение: Суббота, 14 мая 2022, 22:05

◄ 5.2. Видеолекция: Методы обработки естественного языка для мобильных приложений на основе моделей нейронных сетей

Перейти на...

5.4. Практическое занятие 3. Разработка программы-собеседника на естественном языке для мобильных устройств ►