Практическое задание 2.3.5. Создание чат-бота

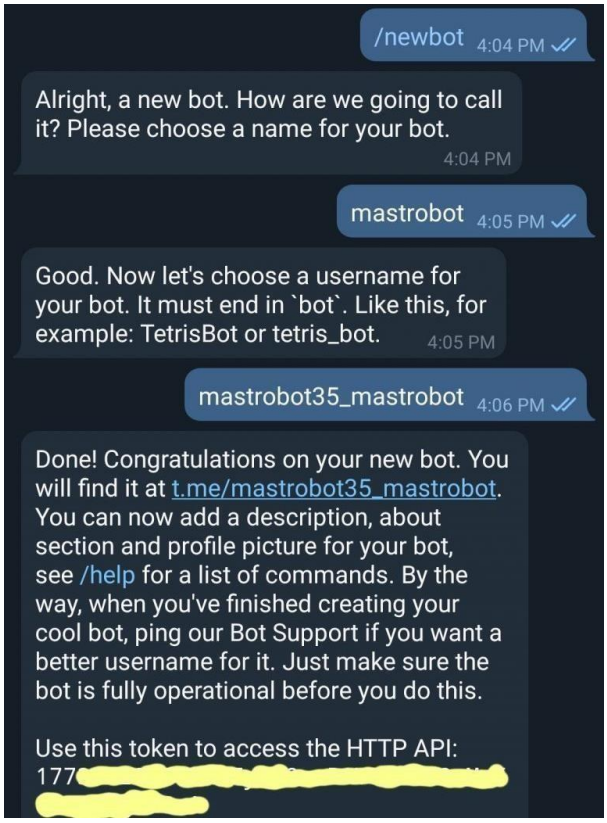
Создать поискового чат-бота для мессенджера Telegram, используя методы NLP. Для этого выполните следующие шаги:

Шаг 1

Зарегистрируйте своего бота на канале @BotFather и получите ссылку и API токен вашего чат-бота.

Комментарии и подсказка

1) Зайдите в приложение telegram. В поисковой строке (знак лупы в верхнем правом углу) напишите @BotFather. В найденных результатах найдите бота-регистратора и начните общение с ним. Обратите внимание, что нужный нам отмечен галочкой!



Отправьте последовательно боту две команды: /start, /newbot. После придумайте имя своему боту и имя пользователя. Сохраните предоставленную ссылку на чат с вашим чат-ботом и API токен.

Шаг 2

Найдите научную статью на определенную тему, изучите и подготовьте её для помещения в корпус. Для этого удалить различные спецзнаки, ссылки на

другие источники и т.д. Сохраните их в разные файлы формата .txt. Для более быстрой работы объем корпуса должен составлять не более 200-250 слов.

Пример:

При Екатерине Великой границы Российской империи были значительно раздвинуты на запад (разделы Речи Посполитой) и на юг (присоединение Новороссии, Крыма, отчасти Кавказа).
Система государственного управления при Екатерине Второй впервые со времени Петра I была реформирована. Реформы Екатерины II подготовили трансформацию русского государства и общества в первой четверти XIX века и стали необходимым условием для реформ 1860-х годов.

Комментарии и подсказка

Если на данном этапе у вас возникли проблемы, то можете воспользоваться следующим фрагментом:

София Фредерика Августа Ангальт-Цербстская родилась 21 апреля (2 мая) 1729 года в немецком городе Штеттин — столице Померании (ныне — Щецин, Польша), в доме № 791 на Домштрассе.

Отец Кристиан Август Ангальт-Цербстский происходил из цербст-дорнбургской линии Ангальтского дома и состоял на службе у прусского короля, был полковым командиром, комендантом, затем губернатором города Штеттина, где будущая императрица и появилась на свет, баллотировался в курляндские герцоги, но неудачно, службу закончил прусским фельдмаршалом. Мать — Иоганна Елизавета, из Готторпского владетельного дома, четвёртая дочь князя Гольштейн-Готторпского, после смерти отца воспитывалась при дворе своего дяди, владетельного князя Брауншвейга. Приходилась двоюродной тёткой будущему Петру III. Родословная Иоганны Елизаветы восходит к Кристиану I, королю Дании, Норвегии и Швеции, первому герцогу Шлезвиг-Гольштейнскому и основателю династии Ольденбургов.

Дядя по материнской линии Адольф-Фридрих был в 1743 году избран в наследники шведского престола, на который он вступил в 1751 году под именем Адольфа-Фредрика. Другой дядя, Карл Эйтинский, по замыслу Екатерины I, должен был стать мужем её дочери Елизаветы, однако умер от оспы в преддверии свадебных торжеств в Санкт-Петербурге.

В семье герцога Цербстского Екатерина получила домашнее образование. Обучалась английскому, французскому и итальянскому языкам, танцам, музыке, основам истории, географии, богословия. Она росла резвой,

любознательной, шаловливой девчонкой, любила щегольнуть своей отвагой перед мальчишками, с которыми запросто играла на штеттинских улицах. Родители были недовольны «мальчишеским» поведением дочери, но их устраивало, что Фредерика заботилась о младшей сестре Августе. Мать называла её в детстве Фике или Фикхен.

Шаг 3

Подключите все необходимые функции и библиотеки.

Комментарии и подсказка

Внимание, не начинайте выполнение работы в Google Collab! В нем telegram боты не поддерживаются! Выполняйте задание в оффлайн редакторах по типу Python IDLE, Geany и другие. Либо же используйте этот онлайн редактор: https://replit.com/ – он самостоятельно установит необходимые расширения при необходимости и поддерживает библиотеки различных ботов, но не выдерживает большие объемы работы, требующие серьезных вычислений.

Все необходимые библиотеки:

```
import string
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
nltk.download('punkt')
try:
     import natasha as nt
except:
     !pip install natasha
     import natasha as nt

try:
     from aiogram import Bot, Dispatcher, executor, types
except:
     !pip install aiogram
     from aiogram import Bot, Dispatcher, executor, types
```

Шаг 4

Создайте вспомогательные функции нормализации текста – Normalize, которая в качестве своего параметра принимает текст из корпуса, и формирование ответа на запрос – Response, которая принимает вопрос от пользователя и формирует ответ на него ссылаясь на информацию из своего корпуса.

Комментарии и подсказка

Данные две функции имеются в лекции №2.3.5 уже в готовом виде:

```
#.....................Вспомогательные функции.....................
#Функция нормализации текста
def Normalize(text):
  #Инициализируем вспомогательные объекты библиотеки natasha
  segmenter = nt.Segmenter()
  morph_vocab = nt.MorphVocab()
  emb = nt.NewsEmbedding()
  morph_tagger = nt.NewsMorphTagger(emb)
  ner_tagger = nt.NewsNERTagger(emb)

  #Убираем знаки пунктуации из текста
  word_token = text.translate(str.maketrans("", "", string.punct
uation)).replace("—", "")

  #Преобразуем очищенный текст в объект Doc и
  doc = nt.Doc(word_token)
  doc.segment(segmenter)
  doc.tag_morph(morph_tagger)
  doc.tag_ner(ner_tagger)

  #Приводим каждое слово к его изначальной форме
  for token in doc.tokens:
    token.lemmatize(morph_vocab)
  resDict = {_.text: _.lemma for _ in doc.tokens}

  #Возвращаем результат в виде списка
  return [resDict[i] for i in resDict]

#Функция ответа на запрос
def Response(user_response):
    user_response = user_response.lower()
    robo_response=''#Будущий ответ нашего бота
    sent_tokens.append(user_response)#Временно добавим запрос по
льзователя в наш корпус.
    TfidfVec = TfidfVectorizer(tokenizer=Normalize)#Вызовем вект
оризатор TF-IDF
    tfidf = TfidfVec.fit_transform(sent_tokens)#Создадим вектора
    vals = cosine_similarity(tfidf[-
1], tfidf)#Через метод косинусного сходства найдем предложение с
 наилучшим результатом
    idx=vals.argsort()[0][-2]#Запомним индексы этого предложения
    flat = vals.flatten()#сглаживаем полученное косинусное сходс
тво
    flat.sort()
    req_tfidf = flat[-2]
    sent_tokens.remove(user_response)
    if(req_tfidf==0): #Если сглаженное значение будет равно 0, т
о ответ не был найден
        robo_response=robo_response+"Извините, я не нашел ответа
..."
        return robo_response
    else:
```

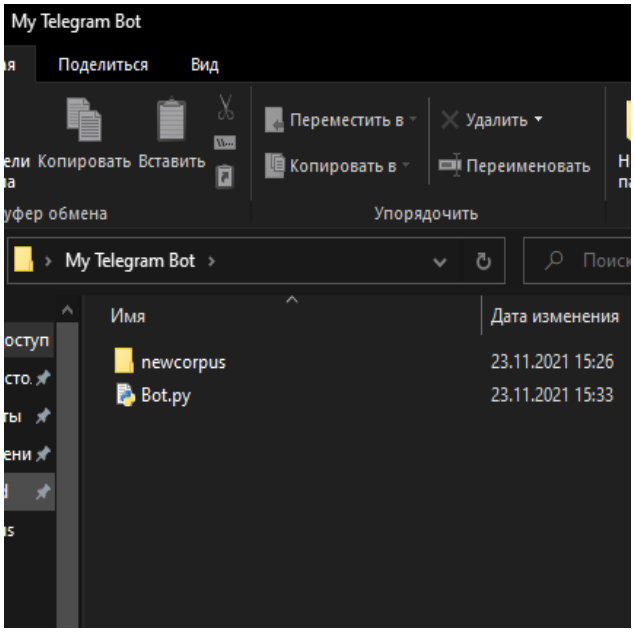
```
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

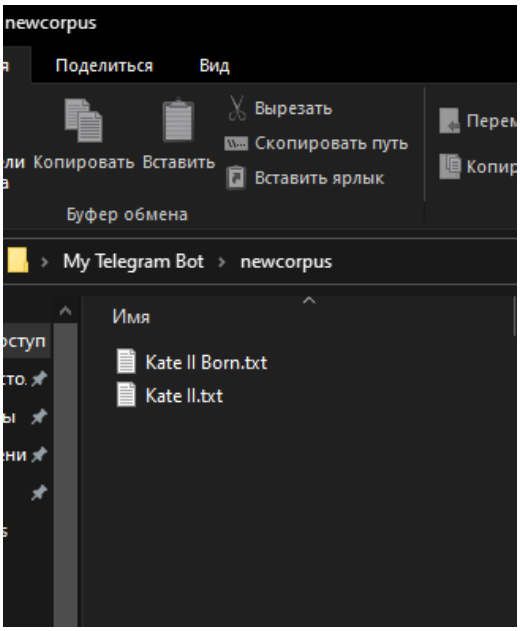
Шаг 5

В основном коде программы (вне функциях) сформируйте корпус из подготовленных заранее txt файлов (сохраните их рядом с основным ру-файлом в отдельной папке newcorpus) и разбейте весь текст на предложения. Сформируйте два списка, которые будут хранить возможные приветствия и прощания пользователя.

Комментарии и подсказка

Вот в таком виде у вас должны хранится основной ру-файл и тексты для корпуса.





Через регулярные выражения найдите необходимые для корпуса файлы и сформируйте сам корпус, разбив его после на предложения:

```
#.....Бот.....

newcorpus = PlaintextCorpusReader('newcorpus/', r'.*\.txt')
#Задание корпуса

data = newcorpus.raw(newcorpus.fileids())
sent_tokens = nltk.sent_tokenize(data)

welcome_input = ["привет", "ку", "прив", "добрый день", "доброго
 времени суток","здравствуйте", "приветствую"]#Список привествий
 goodbye_input = ["пока", "стоп", "выход", "конец", "до свидания"
]#Список прощаний
```

Не забудьте создать списки со словами приветствия и прощания.

Шаг 6

Инициализируйте бота, создав экземпляр класса `Bot` из библиотеки `aiogram` и создайте для него диспетчер – `Dispatcher` (пример есть в лекции). Напишите две асинхронные функции, используя декоратор. Одна из них будет функционировать по ключевому слову «`start`» и будет показывать готовность бота к работе. Вторая функция считывает все остальные сообщения пользователя и определяет содержать ли они слова приветствия и прощания. Если да, то бот должен написать в ответ «Привет!» и «Буду ждать вас!» соответственно. В ином случае функция начинает поиск по корпусу. В конце вызвать команду для запуска бота.

Комментарии и подсказка

Если на данном этапе у вас возникли проблемы, то ниже есть необходимый код. Здесь в строке, где идет инициализация бота, в параметре `token` нужно прописать ваш API токен.

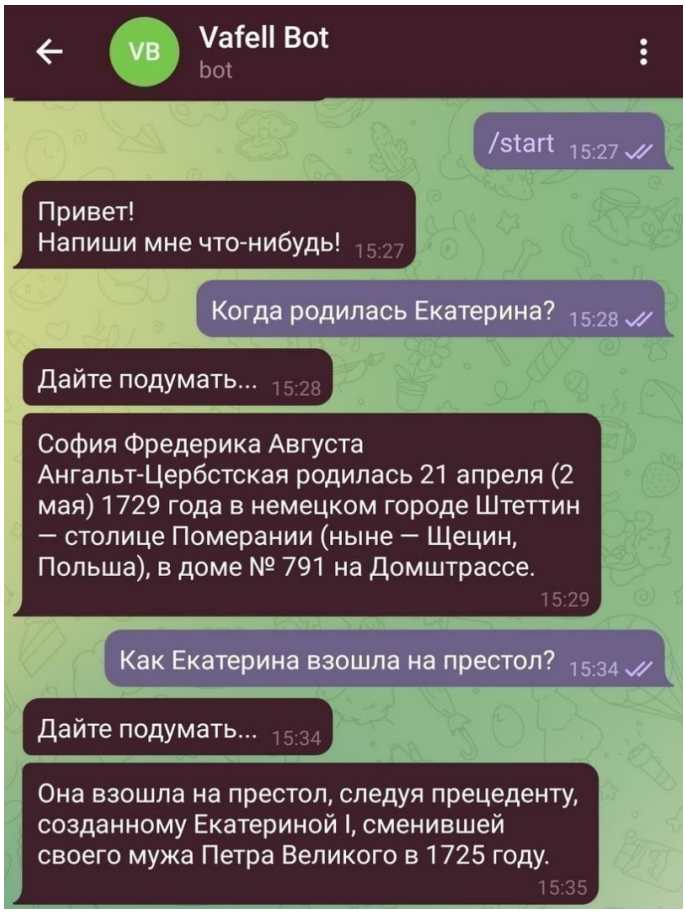
```
bot = Bot(token = 'ВАШ ТОКЕН')#Инициализация бота
dp = Dispatcher(bot)#Определение диспетчера

@dp.message_handler(commands=['start'])#Хэндлер для функции star
t
async def hi_func(message: types.Message):
    await message.answer("Привет!\nНапиши мне что-нибудь!")

@dp.message_handler()#Хэндлер для функции считывания
async def search_func(message: types.message):
  if (message.text).lower() in welcome_input:
    await message.answer('Привет!')
  elif (message.text).lower() in goodbye_input:
    await message.answer('Буду ждать вас!')
  else:
    await message.answer('Дайте подумать...')
    await message.answer(Response(message.text))

#Запуск бота
if __name__ == '__main__':
executor.start_polling(dp, skip_updates=True)
```

В качестве ответа на задание прикрепите WORD-файл, в который вставьте скриншоты из мессенджера telegram, где наглядно показана работа вашего созданного бота, а также ссылку на самого бота. Пример скриншота:



Если же вы не пользуетесь данным мессенджером, то прикрепите в качестве ответа на задание блокнот с кодом.