ASYSTENT GŁOSOWY

Kompletna biblioteka

Streszczenie

Celem niniejszej pracy było przygotowanie biblioteki umożlwiającej implementację asystentów tekstowych i głosowych. Przygotowując odpowiednie klasy, interfejsy, kończąc na przykładowych zastosowaniach i implementacjach.

Jan Bartula, Artem Buhera

bartula.jan@gmail.com artem.bugera@gmail.com

1 Wstęp

Pierwszym etapem przygotowania do wybranego tematu było zapoznanie się z problemem i możliwościami jego realizacji. Mając na przykładzie gotowe rozwiązania takie jak RASA[1] lub ChatterBot[2] chcieliśmy zaimplementować własną bibliotekę bot-a przy użyciu poznanej wiedzy na przedmiocie Przetwarzanie Języka Naturalnego. Nasz projekt zakładał przygotowanie kilka podstawowych aspektów:

- Implementację logiki bota
    - Pre procesorów – mogących przygotować dodatkowe informacje zachowywane w sesji
    - adapterów logicznych - mogących przetwarzać zapytanie zwracając przy tym confidence
        - Z użyciem wektorów TFIDF
    - adapterów wyjściowych - obsługujących wyjściowy rezultat
- Przetwarzania STT jako jednego z adapterów wejściowych
- Przetwarzanie TTS jako adapter wyjściowy
- Dodanie przykładów użycia

2 Rozwinięcie

Implementację rozpoczęliśmy od przygotowania interfejsów:

2.1 Interfejsy

2.1.1 Odpowiedź

Odpowiedź w naszej implementacji zawiera tylko dwa parametry:

- tekst – proponowaną odpowiedź bot-a
- confidence – umożliwiający wybranie najbardziej trafnej odpowiedzi na dane zapytanie

```
class Response:
    def __init__(self, response_text: str, confidence: float):
        self.response_text: str = response_text
        self.confidence: float = confidence

    def __repr__(self):
        return f"{self.confidence:.2f}:{self.response_text[:32]}..."
```

Listing 1 Interfejs odpowiedzi

2.1.2 Pre procesor

Preprocesor umożliwia zapisanie dodatkowych informacji w sesji w celu uzyskania do nich dostępu w adapterach logicznych

[1] https://rasa.com/
[2] https://chatterbot.readthedocs.io/

```python
class PreProcessorAdapter(ABC):

    @abstractmethod
    def process(self, input_text: str, session: dict):
        pass
```

Listing 2 Interfejs Preprocesora

2.1.3 Adapter logiczny

Interfejs adapteru logiczny składa się z dwóch funkcji: can_process i process. Pierwsza odpowiedzialna jest za identyfikację czy tekst spełnia kryterium obsługi przez ten adapter. Druga za właściwe przetworzenie zapytania i zwrócenie odpowiedz. Jest to częściowo przygotowane na styl implementacji w ChatterBot. Ma to jednak swoje uzasadnienie. Jeżeli przyjmiemy że metoda can_process ma mniejszą złożoność obliczeniową od funkcji process. To przy użyciu „eliminacji wstępnej" jesteśmy w stanie zaoszczędzić zużycie niepotrzebnych zasobów systemowych.

```python
class LogicAdapter(ABC):

    @abstractmethod
    def can_process(self, input_text) -> bool:
        return True

    @abstractmethod
    def process(self, input_text: str) -> Response:
        pass
```

Listing 3 Interfejs adaptera logicznego

2.1.4 Rdzeń

Ma za zadanie:

- przyjęcie wypowiedzi,
- przepuszczenie jej przez interfejsy wejściowe
- wybranie adapterów logicznych które mają możliwość przetworzenia informacji
- przetworzenie informacji przez każdy dostępny adapter
- wybranie rezultatu od adaptera z największym confidence
- przekazanie rezultatu do wszystkich interfejsów wyjściowych

```python
class CoreBot:

    def __init__(self) -> None:
        super().__init__()
        self.logic_adapters: List[LogicAdapter] = []
        self.output_adapters: List[Stream] = []
        self.pre_processors: List[PreProcessorAdapter] = []
        self.session = {}

    def add_logic_adapters(self, logic_adapters: List[LogicAdapter]):
        self.logic_adapters.extend(logic_adapters)

    def add_output_adapters(self, stream_adapters: List[Stream]):
        self.output_adapters.extend(stream_adapters)

    def add_pre_processors(self, pre_processors: List[PreProcessorAdapter]):
        self.pre_processors.extend(pre_processors)

    def ask(self, input_text: str):...

    def clean_sessions(self):
        self.session = {}
```

Listing 4 Implementacja rdzenia bot-a

```python
    def ask(self, input_text: str):
        module_logger.info('\t\tBEGIN OF UTTERANCE')
        module_logger.info(f"Asked: {input_text}")
        available_responses = []

        for processor in self.pre_processors:
            processor.process(input_text, self.session)

        module_logger.info("Session: " + str(self.session))

        for adapter in self.logic_adapters:
            if adapter.can_process(input_text, self.session):
                response = adapter.process(input_text, self.session)
                module_logger.debug(f"New Response: {response}")
                available_responses.append(response)

        if len(available_responses) == 0:
            return ""

        match = np.array([res.confidence for res in available_responses]).argsort()[-1]
        module_logger.info(f"Best match: {available_responses[match].response_text}")

        for adapter in self.output_adapters:
            adapter.handle(available_responses[match])
        module_logger.info('\t\tEND OF UTTERANCE\n')
```

Listing 5 Funkcja ask - odpowiedzialna za główną część przetwarzania

2.2 Pre procesor

Zaimplementowaliśmy przykładowy preprocesor umożliwiający ekstrakcję entities przy użyciu fuzzy search zapisując je w sesji rozmowy. Możemy zdefiniować np. kolory, miasta lub inne potrzebne zadania.

```
class EntityExtractorProcessor(PreProcessorAdapter):
    keywords = {}

    def __init__(self, entities: dict):
        self.keywords = entities

    def process(self, input_text: str, session: dict):
        for token in nltk.tokenize.casual_tokenize(input_text):
            for key, entities in self.keywords.items():
                for entity in entities:
                    if len(find_near_matches(entity.lower(), token.lower(), max_l_dist=round(len(entity) / 8))) > 0:
                        session[key] = entity
```

Listing 6 Przykład preprocesor-a extractor entities

```
EntityExtractorProcessor({
    'color': ['red', 'white', 'orange', 'blue'],
    'animal': ['cat', 'dog', 'tiger', 'elephant'],
    'city': ['cracow', 'warsaw', 'oslo', 'new york']
})
```

Listing 7 Przykładowe użycie preprocesora ekstrakcji entities

2.3 Adaptery logiczne

W naszym projekcie zaimplementowaliśmy dwa adaptery logiczne: pierwszy oparty na wyrażeniach regularnych drugi, na wektorach TFiDF

2.3.1 Wyrażenia regularne

Adapter logiczny wyrażeń regularnych ma za zadanie udzielenie odpowiedzi na znaleziony kontekst w zdaniu oraz wskazać pewność odpowiedzi, jeżeli zostaną znalezione podane słowa kluczowe. W tym wypadku implementacja pewności polega na obliczeniu jej ze wzoru:

$$confidence = \text{index}_{\text{match}} * 0.4 + index_{keyword} * 0.6$$

Pewność jest równa średniej ważonej procentowej ilości wyrazów dopasowanych wrażeniem regularnym i wskaźnikiem czy został znaleziony choć jedno słowo kluczowe z wagami 40% do 60%. Pozwala to na obniżenie pewności w przypadku bardzo długich wypowiedzi pomimo znalezienia słowa kluczowego. I znaczne podwyższenie pewności w przypadku jego znalezienia.

Możliwość przetwarzania w danym logicznym adapterze zaimplementowaliśmy poprzez sprawdzenie czy zostało odnalezione dopasowanie w wyrażeniu regularnym.

```python
class RegexLogicAdapter(LogicAdapter, metaclass=ABCMeta):
    @property
    @abstractmethod
    def pattern(self) -> re.Pattern:
        pass

    @property
    @abstractmethod
    def keywords(self) -> List[str]:
        pass

    def can_process(self, statement: str):
        return self.pattern.findall(statement)

    @abstractmethod
    def process(self, input_text: str) -> Response:
        pass

    def calculate_confidence(self, match: str, input_statement: str) -> float:
        match_index = len(match) / len(input_statement)
        keyword_index = self._contains_keyword(input_statement)
        return match_index * 0.4 + keyword_index * 0.6

    def _contains_keyword(self, input_statement: str):
        return any(
            keyword in [
                word.lower()
                for word in
                input_statement.split()
            ]
            for keyword
            in self.keywords
        )
```

Listing 8 Adapter logiczny oparty na wyrażeniach regularnych

2.3.2 Wektory TFiDF

W przypadku adaptera opartego na wektorach TFiDF nie mamy możliwości zrobienia przeszukiwania wstępnego. Teoretycznie można by było sprawdzać czy którykolwiek wyraz znajdzie się w worku słów podanego zbioru tekstowego, jednak nie zastosowaliśmy tego.

Korpus w naszej implementacji przyjmuje listę tupli. Tupel składa się z „pytania” i „odpowiedzi”. Przetwarzanie składa się z przygotowanie zbioru pytań poszerzonego o zapytanie od użytkownika. Przetwarzamy otrzymaną listę przez wektoryzator TFiDF następnie liczmy odległość cosinusową między ostatnim elementem a całą listą wektorów. Po posortowaniu listy od największego dopasowania do najmniejszego pierwszy element listy będzie dodanym wcześniej zapytaniem użytkownika, który pomijamy. Interesują nas 2 i każdy następny element listy. My zwracamy tylko drugi element listy i zwracamy przyporządkowaną do jego indeksu odpowiedź. Tak znajdujemy najbardziej trafną odpowiedź do podanego przez użytkownika zapytania. Pewność w tym przypadku będzie odległością cosinusową pytania użytkownika do znalezionego pytania w zbiorze tekstowym.

W tym przykładzie rozszerzenie zbioru o jedno dodatkowe pytanie od użytkownika znacznie zmniejsza ilość przeprowadzanych operacji, niżeli w przypadku, kiedy obliczalibyśmy wektor TFiDF dla zbioru tekstowego a później dla pojedynczej wypowiedzi użytkownika.

```python
class CorpusLogicAdapter(LogicAdapter):

    def __init__(self, question_answer: list[list[str]]) -> None:
        super().__init__()
        self.question_answer = question_answer

    def can_process(self, input_text) -> bool:
        return True

    def process(self, input_text: str) -> Response:
        corpus = list(map(itemgetter(0), self.question_answer))
        answer = list(map(itemgetter(1), self.question_answer))
        corpus.append(input_text)
        tfidf_vec = TfidfVectorizer(stop_words=stop_words)
        tfidf = tfidf_vec.fit_transform(corpus)
        similarity = cosine_similarity(tfidf[-1], tfidf)
        # print(similarity)
        idx = similarity.argsort()[0][-2]
        return Response(answer[idx], similarity[0][idx])
```

2.4 Adapter zamiany binarnej

```python
class BinaryConvertRegexLogicAdapter(RegexLogicAdapter):
    pattern = re.compile(r'([01]{2,})', flags=re.IGNORECASE)
    keywords = ['binary', 'bin']

    def __init__(self):
        super().__init__()

    def process(self, input_text: str) -> Response:
        binary = self.pattern.findall(input_text)[0]
        text = f'Decimal value of {binary} is {int(binary, 2)}'
        conf = self.calculate_confidence(binary, input_text)
        return Response(text, conf)
```

2.5 Adapter niskiego confidence

Został zaimplementowany przykład użycia adaptera logicznego do obsługi jeżeli inny adapter nie będzie miał zadanej pewności.

W tym przypadku dodajemy zwykły adapter, którego w konstruktorze podajemy wartość pewności zwracanej w czasie przetwarzania. Jedną lub wiele odpowiedzi, z której jedna będzie losowana. W przypadku gdy inne adaptery będą miały niższą pewność od tej podanej w parametrze, mamy pewność, że nie zwrócimy ich odpowiedzi.

```python
class LowConfidenceAdapter(LogicAdapter, metaclass=ABCMeta):

    def __init__(self, confidence: float, response: Union[str, list[str]]) -> None:
        super().__init__()
        self.confidence = confidence

        if isinstance(response, list):
            self.responses = response
        else:
            self.responses = [response]

    def can_process(self, input_text) -> bool:
        return True

    def process(self, input_text: str) -> Response:
        return Response(random.choice(self.responses), self.confidence)
```

3 Przetwarzanie audio

Do przetwarzania dźwięku szukaliśmy nowego rozwiązania przetwarzania mowy na tekst i tekstu na mowę. Jednym z najbardziej aktywnych w przeciągu ostatnich lat repozytorium znaleźliśmy rozwiązania Coqui[3] jest to organizacja założona z projektu Mozilli prowadzącej projekt DeepSpeech oferująca narzędzie STT. Projekt został częściowo zamknięty i został on przejęty przez Coqui. Dodatkowo rozszerzyli narzędzie o przetwarzanie tekstu na mowę.

Jedną z ciekawych oferowanych możliwości TTS są modele wielojęzykowe i wielogłosowe.

3.1 TTS

W ramach implementacji Coqui TTS, napisaliśmy implementację naszego interfejsu przyjmującego wypowiedź od bota, przepuszczając przez jeden z dostępnych modeli. Ponieważ w przypadku niektórych modeli generowanie audio trwa kilka sekund, zaimplementowaliśmy zapisywanie generowanego audio jako pliki dźwiękowe z nazwą wygenerowaną jako hasz tekstu wejściowego.

Dodatkowo w celach polepszenia wypowiedzi usuwamy wszystkie znaki poza literami, cyframi i podstawowymi znakami interpunkcyjnymi.

Odtwarzanie audio polega na otwarciu znalezionego pliku audio z dodatkową sekundową ciszą przed jego odtworzeniem (dla lepszego odbioru)

```
class CoquiTTSStreamAdapter(Stream):

    def __init__(self):
        self.TEMP_PATH: str = "tts_temp_cq"
        if not os.path.isdir(self.TEMP_PATH):
            os.mkdir(self.TEMP_PATH)
        self.regex_filter = re.compile(r'[^\w.,?!]+')
        self.tts = TTS('tts_models/en/ljspeech/tacotron2-DCA')

    def text_cleanup(self, text: str):
        return self.regex_filter.sub(' ', text)

    def handle(self, output: Response):
        text = self.text_cleanup(output.response_text)
        module_logger.info("CLEANED: " + text)

        message_bytes = text.encode('ascii')
        text_hash = hashlib.sha256(message_bytes)
        filename = f"{self.TEMP_PATH}/{text_hash.hexdigest()}.wav"
        if not os.path.isfile(filename):
            module_logger.info("GENERATING: " + filename)
            self.tts.tts_to_file(text=text,
                                 file_path=filename)
        else:
            module_logger.info("CACHE MATCH: " + filename)

        sound = AudioSegment.silent().append(AudioSegment.from_file(file=filename, format="wav"))
        play(sound)
```

[3] https://github.com/coqui-ai/

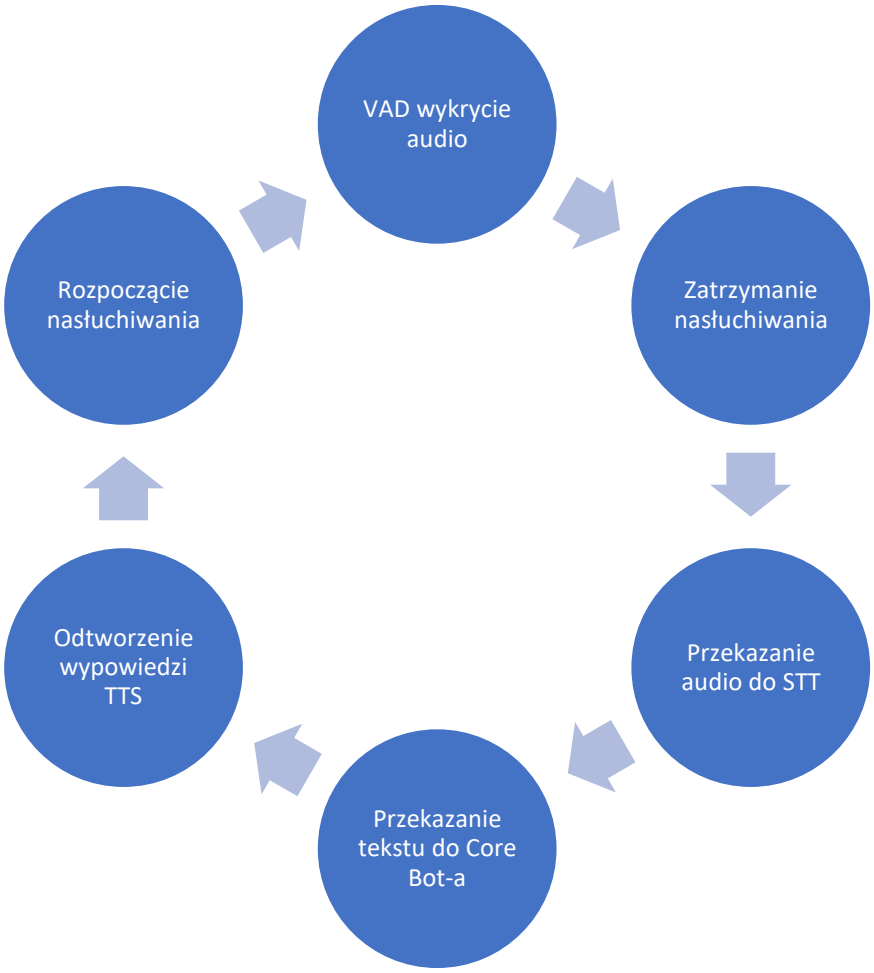
3.2 STT

Część STT jest lekko bardziej skomplikowana. Skorzystaliśmy z dostępnego przykładu dostępnego na repozytorium GitHub[4].

Przetwarzanie składa się z wyboru modelu STT, my użyliśmy największego możliwego wytrenowanego na danych Common Voice 7.0 English[5] - English STT v1.0.0 (Huge Vocabulary).

Następnie podpięcie pod jego strumień wejściowy, strumienia wyjściowego z VAD (Voice Activity Detection) umożliwiającego przekazywanie audio tylko w momencie, kiedy nie jest wykrywana cisza. Pod strumień wejściowy VAD wpięty jest domyślne urządzenie wejściowe komputera.

Przetwarzanie polega na ciągłej operacji:



[4] https://github.com/coqui-ai/STT-examples/blob/r1.0/mic_vad_streaming/README.rst
[5] https://commonvoice.mozilla.org/en/datasets

4 Przykłady użycia

4.1 Pomoc techniczna do drukarek

Jedynym z przykładowych zastosowania zaimplementowanej biblioteki byłby asystent pomocy technicznej do drukarek. Zostały użyte w nim oba logiczne adaptery: wyrażeń regularnych oraz korpusu.

Adapter wykorzystujący wyrażenia regularne został przystosowany do odpowiedzi na pytanie co oznacza podany kod błędu na drukarce na przykładzie drukarek firmy Brother[6]:

```
class ErrorCodeLogicAdapter(RegexLogicAdapter):
    pattern = re.compile(r'#([\da-z]{2,})', flags=re.IGNORECASE)
    keywords = ['error', 'code', '#']

    code_message = {
        '10': 'There is a problem on the duplex unit.',
        '38': 'The machine does not work due to a paper jam.',
        '43': 'The internal temperature is too low or too high.',
        '48': 'There is a problem on the print head.',
        '4F': 'The machine does not work due to a paper jam or some sensor problem.',
        '8F': 'There is a problem on the duplex unit.'
    }

    def __init__(self):
        super().__init__()

    def process(self, input_text: str) -> Response:
        error_code = self.pattern.findall(input_text)[0].upper()
        if error_code in self.code_message:
            return Response(self.code_message[error_code], 1)
        else:
            return Response("Unknown error code", 0)
```

Dodatkowo został przygotowany korpus przygotowany na odpowiadanie w temacie problemów związanych z drukarką, odpowiada przykładowymi odpowiedziami FAQ drukarek firmy Brother[7]

```
test_dialog = [
    [
        "Black lines down the page",
        "Wipe the scanner glass strip with a dry lint free soft cloth.\n"
        "Clean the primary corona wire inside the drum unit by sliding the green tab.\n"
        "The drum unit may be damaged. Put in a new drum unit."
    ],
    [
        "White lines across the page",
        "Clean the drum unit\n"
        "The issue may disappear by itself. Print multiple blank pages to clear this issue, especially if the machine has not been used for a long time."
    ],

    [
        "Curled or wavy paper after print",
        "Check the paper type and quality. High temperatures and high humidity will cause paper to curl\n"
        "Choose Reduce Paper Curl mode in the printer driver when you do not use our recommended print media. "
    ],
    [
        "Who are you",
        "I am Brother printer help assistant"
    ]
]
```

Przykładowe testy rozmów:

6 https://support.brother.com/g/b/faqend.aspx?c=au&lang=en&prod=dcp6690cw_eu_as&faqid=faq00000168_005

7 https://support.brother.com/g/b/faqlist.aspx?c=us_ot&lang=en&prod=mfcl5700dw_us&ftype3=1975

```
INFO:core.logic:        BEGIN OF UTTERANCE
INFO:core.logic:Asked: Print's curled paper
INFO:core.logic:Best match: Check the paper type and quality. High temperatures and high humidity will cause paper to curl
Choose Reduce Paper Curl mode in the printer driver when you do not use our recommended print media.
INFO:core.logic:        END OF UTTERANCE

INFO:core.logic:        BEGIN OF UTTERANCE
INFO:core.logic:Asked: I have error code #8f
INFO:core.logic:Best match: There is a problem on the duplex unit.
INFO:core.logic:        END OF UTTERANCE

INFO:core.logic:        BEGIN OF UTTERANCE
INFO:core.logic:Asked: I have #48 error code what can I do
INFO:core.logic:Best match: There is a problem on the print head.
INFO:core.logic:        END OF UTTERANCE
```

5 Podsumowanie

Przygotowana przez nas biblioteka udostępnia przygotowane interfejsy wraz z ich podstawowymi implementacjami. Kod został przygotowany do elastycznej rozbudowy aplikacji. Wystarczy użyć dostępnych interfejsów w celu implementacji swoich własnych logik biznesowych potrzebnych przy rozwiazywaniu obranych problemów.