Contents

1 Rasa 详细配置和使用指南

1.1 1. Rasa 简介

Rasa 是一个开源的对话 AI 框架，专门用于构建智能聊天机器人和语音助手。它由两个主要组件组成：

1.1.1 1.1 Rasa NLU (自然语言理解)

- **功能**：理解用户输入的文本，提取意图和实体
- **核心任务**：
  - 意图分类 (Intent Classification)
  - 实体提取 (Entity Extraction)
  - 响应选择 (Response Selection)

1.1.2 1.2 Rasa Core (对话管理)

- **功能**：管理对话流程，决定机器人的下一步动作
- **核心任务**：
  - 对话状态跟踪 (Dialogue State Tracking)
  - 策略学习 (Policy Learning)

– 动作预测 (Action Prediction)

1.2 2. Rasa 3.6.21 新特性

1.2.1 2.1 主要改进

- **DIET 分类器优化**：更好的多语言支持
- **GPU 加速增强**：更高效的训练性能
- **内存优化**：减少训练时的内存占用
- **API 稳定性**：更稳定的 HTTP API

1.2.2 2.2 中文支持增强

- 改进的中文分词支持
- 更好的中文实体识别
- 优化的中文语言模型集成

1.3 3. 配置文件详解

1.3.1 3.1 config.yml - 核心配置

```yaml
# 配方版本 - 定义了默认的组件配置
recipe: default.v1

# 语言设置
language: zh  # 中文

# NLU 管道 - 定义文本处理流程
pipeline:
  # 1. 分词器
  - name: WhitespaceTokenizer
    # 基于空格的分词器，处理标点和英文

  - name: JiebaTokenizer
    # 中文分词器，必须安装 jieba
    # pip install jieba
    dictionary_path: null  # 可选：自定义词典路径

  # 2. 特征提取器
  - name: RegexFeaturizer
    # 正则表达式特征提取
    # 用于识别电话、邮箱等模式

  - name: LexicalSyntacticFeaturizer
    # 词汇语法特征提取
    # 提取词性、词形等特征

  - name: CountVectorsFeaturizer
    # 词频向量特征提取
    analyzer: char_wb      # 字符级 n-gram，适合中文
    min_ngram: 1           # 最小 n-gram 长度
    max_ngram: 4           # 最大 n-gram 长度

  # 3. 深度学习组件
  - name: LanguageModelFeaturizer
    # 预训练语言模型特征提取
    model_name: "bert"
    model_weights: "rasa/bert-base-chinese"
    # 注意：需要下载中文 BERT 模型
```

```yaml
  - name: DIETClassifier
    # 双重意图实体转换器 - 核心分类器
    epochs: 100                     # 训练轮次
    constrain_resources: false      # 允许使用所有资源（GPU）
    entity_recognition: true        # 启用实体识别
    intent_classification: true     # 启用意图分类
    use_masked_language_model: true # 使用掩码语言模型

    # 模型架构配置
    hidden_layers_sizes:
      text: [256, 128]              # 文本特征层
      label: [256, 128]             # 标签特征层

    # 训练参数
    batch_size: [64, 256]           # 批次大小范围
    learning_rate: 0.001            # 学习率
    drop_rate: 0.2                  # Dropout 率

  # 4. 后处理组件
  - name: EntitySynonymMapper
    # 实体同义词映射

  - name: ResponseSelector
    # 响应选择器，用于 FAQ 和闲聊
    epochs: 100
    constrain_resources: false

  - name: FallbackClassifier
    # 兜底分类器
    threshold: 0.3              # 置信度阈值
    ambiguity_threshold: 0.1    # 歧义阈值

# 对话管理策略
policies:
  - name: MemoizationPolicy
    # 记忆策略 - 记住训练故事
    max_history: 5

  - name: RulePolicy
    # 规则策略 - 处理固定规则

  - name: UnexpecTEDIntentPolicy
    # 意外意图策略
    max_history: 5
    epochs: 100
    constrain_resources: false

  - name: TEDPolicy
    # TED 策略 - 主要对话管理策略
    max_history: 5
    epochs: 100
    constrain_resources: false

# 助手 ID
assistant_id: instruction_training_platform
```

1.3.2 3.2 domain.yml - 领域定义

```yaml
version: "3.1"

# 意图定义 - 用户可能表达的意图
intents:
  - greet                    # 问候
  - goodbye                  # 告别
  - affirm                   # 确认
  - deny                     # 否认
  - mood_great               # 心情好
  - mood_unhappy             # 心情不好
  - bot_challenge            # 询问是否是机器人
  - ask_weather              # 询问天气
  - book_flight              # 预订航班
  - cancel_booking           # 取消预订
  - ask_help                 # 寻求帮助

# 实体定义 - 可提取的信息片段
entities:
  - city                     # 城市
  - date                     # 日期
  - time                     # 时间
  - person_name              # 人名
  - booking_id               # 预订 ID

# 槽位定义 - 存储对话信息
slots:
  departure_city:
    type: text
    influence_conversation: true
    mappings:
    - type: from_entity
      entity: city

  arrival_city:
    type: text
    influence_conversation: true
    mappings:
    - type: from_entity
      entity: city

# 响应模板 - 机器人回复
responses:
  utter_greet:
    - text: " 您好！我是智能助手，有什么可以帮您的吗？"
    - text: " 你好！欢迎使用我们的服务，请问需要什么帮助？"

  utter_goodbye:
    - text: " 再见！祝您生活愉快！"
    - text: " 谢谢使用我们的服务，再见！"

# 表单定义 - 收集用户信息
forms:
  flight_booking_form:
    required_slots:
      - departure_city
      - arrival_city
      - travel_date
```

```
      - passenger_name

# 动作定义
actions:
  - action_book_flight
  - action_cancel_booking
  - utter_greet
  - utter_goodbye

# 会话配置
session_config:
  session_expiration_time: 60    # 会话过期时间（分钟）
  carry_over_slots_to_new_session: true
```

1.4 4. 训练数据格式

1.4.1 4.1 NLU 训练数据 (data/nlu.yml)

```
version: "3.1"

nlu:
# 问候意图
- intent: greet
  examples: |
    - 你好
    - 您好
    - 早上好
    - 下午好
    - 晚上好
    - 嗨
    - hello
    - hi

# 预订航班意图（包含实体）
- intent: book_flight
  examples: |
    - 我想预订航班
    - 帮我订机票
    - 我要买机票
    - 预订从 [北京](city) 到 [上海](city) 的航班
    - 我想订 [明天](date) 的机票
    - 帮我查一下 [12 月 25 日](date) 的航班
    - 我要从 [广州](city) 飞 [深圳](city)
    - 预订 [下周一](date) 的机票

# 同义词定义
- synonym: 北京
  examples: |
    - 北京市
    - 首都
    - BJ

- synonym: 上海
  examples: |
    - 上海市
    - 魔都
    - SH

# 正则表达式实体
```

```
- regex: phone_number
  examples: |
    - \d{3}-\d{4}-\d{4}
    - \d{11}

# 查找表实体
- lookup: city
  examples: |
    - 北京
    - 上海
    - 广州
    - 深圳
    - 杭州
```

1.4.2 4.2 故事数据 (data/stories.yml)

```
version: "3.1"

stories:
# 基本问候故事
- story: 简单问候
  steps:
  - intent: greet
  - action: utter_greet

# 航班预订故事
- story: 预订航班流程
  steps:
  - intent: book_flight
  - action: utter_book_flight
  - action: flight_booking_form
  - active_loop: flight_booking_form
  - slot_was_set:
    - requested_slot: departure_city
  - intent: inform
    entities:
    - city: " 北京"
  - slot_was_set:
    - departure_city: " 北京"
  - slot_was_set:
    - requested_slot: arrival_city
  - intent: inform
    entities:
    - city: " 上海"
  - slot_was_set:
    - arrival_city: " 上海"
  - slot_was_set:
    - requested_slot: null
  - active_loop: null
  - action: action_book_flight
```

1.4.3 4.3 规则数据 (data/rules.yml)

```
version: "3.1"

rules:
# Fallback 规则
- rule: 激活 fallback
```

```yaml
  steps:
  - intent: nlu_fallback
  - action: utter_fallback

# 表单激活规则
- rule: 激活航班预订表单
  condition:
  - active_loop: null
  steps:
  - intent: book_flight
  - action: flight_booking_form
  - active_loop: flight_booking_form

# 基本响应规则
- rule: 回应问候
  steps:
  - intent: greet
  - action: utter_greet
```

1.5 5. 自定义动作开发

1.5.1 5.1 动作服务器 (actions/actions.py)

```python
from typing import Any, Text, Dict, List
from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.forms import FormAction
import logging

logger = logging.getLogger(__name__)

class ActionBookFlight(Action):
    """ 航班预订动作"""

    def name(self) -> Text:
        return "action_book_flight"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        # 获取槽位值
        departure_city = tracker.get_slot("departure_city")
        arrival_city = tracker.get_slot("arrival_city")
        travel_date = tracker.get_slot("travel_date")
        passenger_name = tracker.get_slot("passenger_name")

        # 验证必要信息
        if not all([departure_city, arrival_city, travel_date, passenger_name]):
            dispatcher.utter_message(text=" 预订信息不完整，请重新提供。")
            return []

        # 模拟预订逻辑
        try:
            # 这里可以调用真实的预订 API
            booking_id = self._create_booking(
                departure_city, arrival_city, travel_date, passenger_name
            )
```

```
            message = f" 预订成功！\\n" \
                      f" 乘客：{passenger_name}\\n" \
                      f" 航线：{departure_city} → {arrival_city}\\n" \
                      f" 日期：{travel_date}\\n" \
                      f" 预订号：{booking_id}"

            dispatcher.utter_message(text=message)

            # 记录预订信息
            logger.info(f" 航班预订成功：{booking_id}")

        except Exception as e:
            logger.error(f" 预订失败：{e}")
            dispatcher.utter_message(text=" 预订失败，请稍后重试。")

        return []

    def _create_booking(self, departure, arrival, date, passenger):
        """ 模拟创建预订"""
        import hashlib
        booking_data = f"{departure}{arrival}{date}{passenger}"
        booking_id = f"FL{hashlib.md5(booking_data.encode()).hexdigest()[:6].upper()}"
        return booking_id

class ValidateFlightBookingForm(FormAction):
    """ 航班预订表单验证"""

    def name(self) -> Text:
        return "validate_flight_booking_form"

    @staticmethod
    def required_slots(tracker: Tracker) -> List[Text]:
        return ["departure_city", "arrival_city", "travel_date", "passenger_name"]

    def validate_departure_city(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> Dict[Text, Any]:
        """ 验证出发城市"""

        valid_cities = [" 北京", " 上海", " 广州", " 深圳", " 杭州", " 南京"]

        if slot_value and slot_value in valid_cities:
            return {"departure_city": slot_value}
        else:
            dispatcher.utter_message(
                text=f" 抱歉，我们暂不支持从 {slot_value} 出发的航班。"
            )
            return {"departure_city": None}

    def validate_travel_date(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
```

```
        domain: Dict[Text, Any],
    ) -> Dict[Text, Any]:
        """ 验证出行日期"""

        # 这里可以添加日期格式验证和业务逻辑验证
        if slot_value:
            # 简单验证：检查是否包含日期关键词
            date_keywords = [" 今天", " 明天", " 后天", " 下周", " 月", " 日"]
            if any(keyword in slot_value for keyword in date_keywords):
                return {"travel_date": slot_value}

        dispatcher.utter_message(text=" 请提供有效的出行日期，如：明天、下周一、12 月 25 日等。")
        return {"travel_date": None}
```

1.6 6. 训练和部署

1.6.1 6.1 训练命令

```
# 训练 NLU 模型
rasa train nlu

# 训练完整模型（NLU + Core）
rasa train

# 使用 GPU 训练
CUDA_VISIBLE_DEVICES=0 rasa train

# 指定配置文件训练
rasa train --config config.yml --domain domain.yml --data data/
```

1.6.2 6.2 测试命令

```
# 测试 NLU
rasa test nlu

# 测试故事
rasa test core

# 交互式测试
rasa shell

# NLU 交互式测试
rasa shell nlu
```

1.6.3 6.3 服务启动

```
# 启动 Rasa 服务器
rasa run --enable-api --cors "*" --port 5005

# 启动动作服务器
rasa run actions --port 5055

# 调试模式启动
rasa run --enable-api --cors "*" --debug
```

1.7 7. API 使用

1.7.1 7.1 预测 API

```
import requests

# 发送预测请求
url = "http://localhost:5005/model/parse"
data = {
    "text": " 我想预订明天从北京到上海的航班"
}

response = requests.post(url, json=data)
result = response.json()

print(" 意图:", result["intent"]["name"])
print(" 置信度:", result["intent"]["confidence"])
print(" 实体:", result["entities"])
```

1.7.2 7.2 对话 API

```
# 发送对话请求
url = "http://localhost:5005/webhooks/rest/webhook"
data = {
    "sender": "user123",
    "message": " 你好"
}

response = requests.post(url, json=data)
messages = response.json()

for message in messages:
    print(" 机器人回复:", message["text"])
```

1.8 8. 性能优化

1.8.1 8.1 GPU 配置

```
# 在训练前设置 GPU
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

# TensorFlow GPU 配置
import tensorflow as tf
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    tf.config.experimental.set_memory_growth(gpus[0], True)
```

1.8.2 8.2 训练优化

```
# config.yml 优化配置
pipeline:
  - name: DIETClassifier
    # 批次大小优化
    batch_size: [64, 256]

    # 早停配置
    evaluate_every_number_of_epochs: 20
```

```
    evaluate_on_number_of_examples: 0

    # 模型压缩
    number_of_transformer_layers: 2
    transformer_size: 256

    # 正则化
    drop_rate: 0.2
    weight_sparsity: 0.8
```

1.8.3 8.3 推理优化

```
# 模型缓存配置
import rasa.core.agent

# 创建代理并缓存模型
agent = rasa.core.agent.Agent.load("models/")

# 批量预测
texts = [" 你好", " 我想订机票", " 再见"]
results = []

for text in texts:
    result = agent.parse_message(text)
    results.append(result)
```

1.9 9. 常见问题

1.9.1 9.1 中文分词问题

```
# 安装 jieba
pip install jieba

# 如果遇到编码问题
export PYTHONIOENCODING=utf-8
```

1.9.2 9.2 GPU 内存不足

```
# 减少批次大小
pipeline:
  - name: DIETClassifier
    batch_size: [32, 128]  # 减小批次大小

    # 减少模型复杂度
    hidden_layers_sizes:
      text: [128, 64]
      label: [128, 64]
```

1.9.3 9.3 训练速度慢

```
# 减少训练轮次
pipeline:
  - name: DIETClassifier
    epochs: 50  # 从 100 减少到 50

policies:
  - name: TEDPolicy
```

```
    epochs: 50
```

1.10 10. 最佳实践

1.10.1 10.1 数据准备

- 每个意图至少准备 10-20 个样本
- 样本要多样化，覆盖不同表达方式
- 实体标注要准确一致
- 定期清理和更新训练数据

1.10.2 10.2 模型调优

- 从小数据集开始，逐步增加
- 使用交叉验证评估模型性能
- 监控过拟合，适当使用正则化
- 定期重新训练模型

1.10.3 10.3 生产部署

- 使用模型版本管理
- 设置模型性能监控
- 准备模型回滚机制
- 建立 A/B 测试流程

通过以上详细的配置和使用指南，您可以充分利用 Rasa 3.6.21 的强大功能，构建高质量的中文对话系统。