Reference

[Rasa Masterclass - episode 2]
[Rasa Masterclass - episode 3]
[Rasa Masterclass - episode 4]

1. rasa 학습 데이터 만들기
2. NLU 모델 정의(processing pipeline 정의)
3. NLU 학습결과 테스트

1. rasa 학습 데이터 만들기

(1) conversation design # 어떤 내용에 대한 학습 데이터를 마련해야 하는지 파악하기 위한 작업

- 어시스턴트를 구축하는 것은 conversation design에서 시작
- conversation design: 자신이 챗봇을 구축하려는 분야에서 사용자가 누구인지, 그들이 무슨 목적으로 어시스턴트를 사용하는지, 그들이 어시스턴트와 나누는 가장 전형적인 대화가 무엇인지 파악하는 것
- 이를 위해 크게 4가지 방법을 사용할 수 있음
  1. 해당 분야 전문가에게 자문
  2. 웹사이트가 있다면, 사용자들이 무엇을 가장 많이 검색하는지 확인
  3. 고객센터가 있다면, 사용자들이 가장 많이하는 질문 리스트 확인
  4. 위의 방법을 사용하는 것이 불가능하다면, wizard-of-oz 패러다임을 이용. 직접 사용자들에게 챗봇인 것처럼 질문하여 정보를 얻음

(2) 학습 데이터 구축

- conversational design을 통해 자신이 챗봇을 구축하려는 분야에서 사용자들이 가장 많이 하는 질문이 무엇인지 파악한 후, 해당 질문들에 대한 학습 데이터를 생성
- 예: Medicare Locator Assistant → greeting, facility they are looking for(e.g., hospital, nursing home), location, goodbye message 에 대한 학습 데이터가 필요
- 개발 초기 단계에서 이러한 example conversation을 통해 학습시키고, 이후 실제 사용자와 대화하게 하며 계속 학습시켜야 함
- `nlu.md` 파일에 학습 데이터를 입력하면 됨
  1. intent: `## intent: intent_name`의 형태로 입력
  2. entity: `[entity_value](entity_name)`의 형태로 입력

```
## intent: infrom
- [Sitka](location)
- [Juneau](location)
- [Virginia](location)
- [Cusseta](location)
- [Chicago](location)
- [Tuscon](location)
- [Columbus](location)
- [San Francisco](location)

## intent: search_provider
- I need a [hospital](facility_type) <!--[entity_value](entity_name의 형태로 entity 설정-->
- find me a nearby [hospital](facility_type)
- show me [home health agencies](facility_type)
- [hospital](facility_type)
- find me a nearby [hospital](facility_type) in [San Francisco](location)
- I need a [home health agency](facility_type)
```

(3) 학습 데이터 format

- 학습데이터는 4가지 구성요소를 지님: (1) common examples (2) synonyms (3) regex features (4) lookup tables.
- common examples이 반드시 필요한 핵심요소. 나머지는 적은 수의 데이터로 학습 데이터를 더 잘 학습할 수 있도록 도와주는 요소

1. common examples

- 3가지 구성요소로 이루어져 있음: text(user message), intent, entities

2. synonym

- entities를 추출하였을 때 실제 텍스트에 나타난 문자 대신 synonym을 entity value로 사용하게 되는 것

a. 학습 데이터의 entities value에 synonym을 입력

- `EntitySynonymMapper` component를 pipeline에 추가해야함. 이때 pipeline에서 entity extraction component 이후에 위치시켜야 함

```
## intent:search
- in the center of [NYC]{"entity": "city", "value": "New York City"}
- in the centre of [New York City](city)
```

b. 학습데이터에 “entity_synonyms” array를 추가하여 정의하는 방법

```
## synonym:New York City
- NYC
- nyc
- the big apple
```

3. regex(regular expression) features

- zipcode, email address와 같이 entity나 intent가 일관적인 형태를 가지고 있을 경우 사용
- entity 관련 regex feature은 `CRFEntityExtractor` component로 구현 가능
  intent 관련 regex feature은 모든 intent classifier이 제공

```
## regex:zipcode
- [0-9]{5}

## regex:greet
- hey[^\\s]*
```

4. lookup tables

- 특정 entities의 예시 리스트를 제공해줌
- table file은 반드시 newline-delimited format 여야함
- 학습 데이터에서 exact match를 찾기 위한 case-insensitive regex pattern을 생성해내는 것

```
tacos
beef
mapo tofu
burrito
lettuce wrap
```

```
## lookup:plates
data/test/lookup_tables/plates.txt

## intent:food_request
- I'd like beef [tacos](plates) and a [burrito](plates)
- How about some [mapo tofu](plates)
```

- Markdown, JSON, single file, a directory containing multiple files 등 다양한 형태로 학습 데이터를 입력할 수 있음. Markdown이 가장 작업하기 편한 포맷

1. Markdown

- `[<entity text>](<entity name>)` 또는 `[<entity-text>]{"entity": "<entity name>"}`의 형태로 표현.
- 2번째 형태의 경우 synonym, role, group 등에 대한 정보를 추가할 수 있음: `[<entity-text>]{"entity": "<entity name>", "role": "<role name>", "group": "<group name>", "value": "<entity synonym>"}`

```
- I want to fly from [Berlin]{"entity": "city", "role": "departure"} to [San Francisco]{"entity": "city", "role": "destination"}.
- Give me a [small]{"entity": "size", "group": "1"} pizza with [mushrooms]{"entity": "topping", "group": "1"} and a [large]{"entity": "size", "group": "2"} [pepperoni]{"entity": "topping", "group": "2"}
```

2. JSON

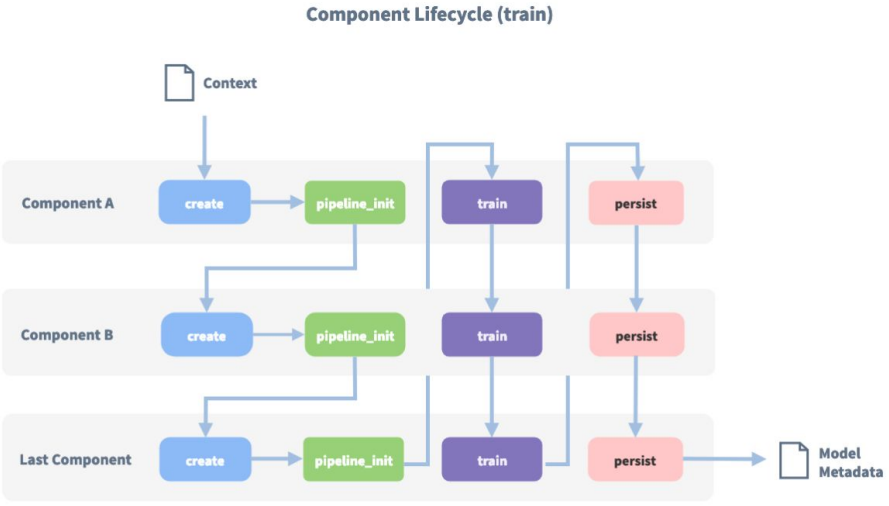
```
{
    "rasa_nlu_data": {
        "common_examples": [],
        "regex_features" : [],
        "lookup_tables"  : [],
        "entity_synonyms": []
    }
}
```

[예시]

(4) 학습 데이터를 구축할 때 유의해야 할 사항

- 하나의 intent 당 10~15개 정도의 예시 데이터로 시작하는 것이 좋음. 예시 데이터는 많을 수록 좋음
- 예시 데이터가 intent와 부합하는지 확인해야 함
- 단어, 문장구조 등의 측면에서 다양한 예시 데이터가 있는 것이 좋음

2. NLU 모델 정의하기; Processing Pipeline 정의하기



(1) processing pipeline

- 학습 데이터에서 특정 text features를 추출하고 이를 기반으로 데이터에 내재된 패턴을 학습하는 일련의 processing step. 이러한 일련의 processing pipeline이 곧 NLU 모델
- 이를 바탕으로 새롭게 주어진 텍스트 메시지의 intent와 entity를 예측

(2) pipeline component [Components] [Custom NLU Components]

- 이전에는 pre-configured pipeline template을 제공하였으나, 지금은 deprecated됨.
  `config.yml` 파일에 사용할 component들을 순서대로 나열시키면 됨

```yaml
# Configuration for Rasa NLU.
# https://rasa.com/docs/rasa/nlu/components/
language: en
pipeline: supervised_embeddings
```

supervised_embeddings

```yaml
language: "en"

pipeline:
- name: "WhitespaceTokenizer"
- name: "RegexFeaturizer"
- name: "CRFEntityExtractor"
- name: "EntitySynonymMapper"
- name: "CountVectorsFeaturizer"
- name: "CountVectorsFeaturizer"
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
- name: "EmbeddingIntentClassifier"
```

pretrained_embeddings_spacy

```yaml
language: "en"

pipeline:
- name: "SpacyNLP"
- name: "SpacyTokenizer"
- name: "SpacyFeaturizer"
- name: "RegexFeaturizer"
- name: "CRFEntityExtractor"
- name: "EntitySynonymMapper"
- name: "SklearnIntentClassifier"
```

↓

```yaml
# Configuration for Rasa NLU.
# https://rasa.com/docs/rasa/nlu/components/
language: en
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
```

- pipeline에서 component를 나열한 순서대로 인풋 메시지가 처리된다. 따라서 component 나열 순서가 중요함
- pipeline은 보통 3가지 main part로 구성됨: (1) tokenization (2) featurization (3) intent classification / entity recognition / response selector

0. word vector source

- pre-trained word embeddings을 불러오는 것 (optional)
- `SpacyNLP`: spacy language model을 불러옴. 이 모델을 사용하려는 경우에만 pipeline에 추가하면 됨

1. tokenization

- 인풋 메시지를 token의 형태로 나누는 과정
- tokenization은 가장 첫 processing step이기 때문에, pipeline에서 제일 상단에 위치해야 함
- `ConveRTTokenizer`: 추천하는 tokenizer. 그러나 영어만 사용 가능
- `WhitespaceTokenizer`: 공백을 기준으로 나뉘는 단어에 사용할 수 있음 - 한국어도 사용가능?
- `SpacyTokenizer`: spacy language model을 사용할 경우

2 entity extraction

- token을 인풋으로 받아 entity recognition result를 산출
- entity recognition의 결과는 NLU 모델 전체 결과에 직접적으로 추가됨: 문장에서 entity가 무엇인지, 어떤 종류의 entity인지. 그 예측에 대한 confidence
- `CRFEntityExtractor`(*CRF; Conditional Random Field)
  - target word와 surrounding word의 text feature(e.g., prefix,suffix)를 통해 entity를 파악
  - POS 태깅을 같이 사용할 수 있음. 이를 위해서 spaCy가 설치되어 있어야 함
- `SpacyEntityExtractor`
  - spacy language model을 사용할 경우
- `DucklingHttpExtractor`
  - number, data 등을 추출하는데 특화됨

- `RegexFeaturizer`
    - more advanced entity extraction을 위해, regular expression이나 lookup tables을 추가하는 것
    - `CRFEntityExtractor` component 앞에 위치시켜야 함

3. intent classification

3.1 featurization

- token으로부터 더 deeper feature를 추출해내는 과정. 이를 기반으로 intent classification model은 텍스트에 내재된 패턴을 파악하고 새로운 텍스트에 대한 예측을 함
- token을 가지고 처리하는 과정이기 때문에 pipeline에서 반드시 tokenizer 뒤에 위치해야 함
- pre-trained word embedding을 제공하는 component를 사용할 지, 제공하지 않는 component를 사용할 지 결정해야 함.
  (*pre-trained word embedding이란: 예를 들어 "I want to buy apples"라는 텍스트 데이터로 부터 "get pears"라는 intent를 예측할 수 있어야함. 이때 모델이 apples과 pears가 유사하다는 것을 미리 알고 있게 되는 것)

1. pre-trained embedding
    - 학습 데이터의 수가 적고, 자신이 다루는 언어에 대한 pre-trained model이 있을 경우
    - `SpacyFeaturizer`
    - `ConveRTFeaturizer`: 영어만 지원
    - `LanguageModelFeaturizer`: BERT, GPT-2 등의 pre-trained 언어 모델을 사용

2. supervised embedding
    - 학습 데이터의 수가 많거나, 자신이 다루는 언어에 대한 pre-trained model이 없거나, domain specific model을 구축하고 싶은 경우
    - `CountVectorsFeaturizer`
        - sklearn의 `CountVectorizer`을 통해 인풋 메시지로부터 BoW representation 형성. 이것이 intent classification model의 인풋으로 사용됨
        - Bag-of-n-grams를 인풋으로 사용하려고 할 때:

```
name: "CountVectorsFeaturizer"
analyzer:"char_wb"
min_ngram: 1
max_ngram: 4
```

3.2 intent classification model

- intent classification의 결과는 NLU 모델 전체 결과에 직접적으로 추가됨

1. supervised embedding을 사용하는 경우
   - `DIETClassifier`(`EmbeddingIntentClassifier`에서 대체됨)이 좋음. `CountVectorFeautrizer`을 같이 사용하는 것을 권장
   - intent classifier는 인풋 메시지와 (학습 데이터의) intent label을 같은 공간에 embed함. supervised embeddings는 둘 사이의 유사성을 최대화하도록 훈련하여 가장 확률이 높은 intent를 도출

   - Multi-intent classification
     - multiple intent(e.g., thank + goodbye), hierarchical intent structure(e.g., feedback + positive / feedback + negative) 를 다루는 경우
     - 1. `DIETClassifier`를 pipeline에 추가
       2. tokenizer에서 flag 정의
          - `intent_tokenization_flag: True` → intent를 여러 token으로 나누고 싶다고 말하고는 것으로, 결국 multiple intent를 다룰 것이라는 것을 의미
          - `intent_split_symbol: "_" or "+"` → intent가 "_" 또는 "+" 기준으로 나누어진다는 것을 의미

```
pipeline:
- name: "CountVectorsFeaturizer"
- name: "EmbeddingIntentClassifier"
  intent_tokenization_flag: true
  intent_split_symbol: "+"
```

```
## intent: affirm+ask_transport
- Yes. How do I get there?
- Sounds good. Do you know how I could get there from home?
```

2. pre-trained word embedding을 사용하는 경우
   - `SklearnIntentClassifier`
   - 위의 classifier은 Support Vector Machine(SVM)를 define하는 것. SVM은 text feature를 기반으로 user input의 intent를 파악
   - `SpacyFeaturizer`과 함께 많이 사용

3. NLU 학습 결과 테스트 해보기

- 학습 데이터를 입력하고, pipeline을 정의한 후 이를 기반으로 NLU train 시키기: `rasa train` or `rasa train nlu` (nlu 모델만 훈련시키는 코드)

- nlu 모델 테스트하기: `rasa shell nlu`
    - nlu 모델이 새로운 텍스트 데이터에 대해 예측한 intent와 entity가 confidence rate와 함께 산출됨. intent의 경우 후보군에 대한 결과도 나옴