Rasa NLU 语言理解工具

注意

这是 Rasa NLU 版本 0.11.4 的文档。确保为本地安装选择适当的文档版本！

Rasa NLU 是用于意图分类和实体提取的开源工具。例如，像一个句子

```
"I am looking for a Mexican restaurant in the center of town"
```

并返回结构化数据

```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine" : "Mexican",
    "location" : "center"
  }
}
```

目标受众主要是发展机器人的人。您可以使用 rasa 作为智慧，LUIS 或 Dialogflow 的替代插件，代码中的唯一更改是向请求发送请求 `localhost`（请参阅迁移现有应用程序以了解详细信息）。

为什么你可以使用 rasa 而不是其中的一种服务？

- 您不必将您的数据交给 FB / MSFT / GOOG
- 你不必 `https` 每次都打电话。
- 您可以调整模型以适合您的特定用例。

这些观点在博客文章中有更详细的介绍。

最快的快速入门

```
$ python setup.py install
$ python -m rasa_nlu.server -e wit &
$ curl 'http://localhost:5000/parse?q=hello'
[{"_text": "hello", "confidence": 1.0, "entities": {}, "intent": "greet"}]
```

你走了！你只是解析了一些文字。下一步，做教程：一个简单的餐厅搜索机器人。

注意

这个演示使用了一个非常有限的 ML 模型。要将 Rasa NLU 应用于您的用例，您需要训练自己的模型！按照教程了解如何将 rasa_nlu 应用于您的数据。

关于

您可以将 Rasa NLU 视为一组使用现有 NLP 和 ML 库构建自己的语言解析器的高级 API。设置过程被设计得尽可能简单。如果您正在使用智慧，LUIS 或 Dialogflow，则只需：

1. 从 wit 或 LUIS 下载您的应用程序数据并将其提供给 Rasa NLU
2. 在你的机器上运行 Rasa NLU，并将 wit / LUIS / Dialogflow API 调用的 URL 切换到 `localhost:5000/parse`。

Rasa NLU 是用 Python 编写的，但它可以通过使用 Rasa NLU 作为 HTTP 服务器从任何语言使用它。如果你的项目*是*用 Python 编写的，你可以简单地导入相关的类。

Rasa 是由 Rasa 开发的一套用于构建更高级机器人的工具。这是自然语言理解模块，也是开源的第一个组件。

入门

- 安装
- 教程：一个简单的餐厅搜索机器人

用户文档

- 配置
- 迁移现有的应用程序
- 训练数据格式
- 使用 Rasa NLU 作为 HTTP 服务器
- 从 python 使用 Rasa NLU
- 实体提取
- 从反馈中改进模型
- 模型持久化
- 语言支持
- 处理管道
- 评估

- 经常问的问题
- 迁移指南
- 许可证

开发者文档

- 路线图
- 贡献
- 更改日志

安装

Rasa NLU 本身没有任何外部依赖，但要做一些有用的事情，你需要安装和配置一个后端。您想使用哪个后端由您决定。

设置 Rasa NLU

推荐安装 Rasa NLU 的方法是使用 pip：

```
pip install rasa_nlu
```

如果你想使用 github + setup.py 使用最新版本的版本：

```
git clone https://github.com/RasaHQ/rasa_nlu.git
cd rasa_nlu
pip install -r requirements.txt
pip install -e .
```

Rasa NLU 允许您使用组件来处理您的消息。例如，存在用于意图分类的组件，并且存在用于实体识别的多个不同组件。不同的组件有他们自己的要求。为了让您快速入门，本安装指南仅安装基本要求，如果您要使用某些组件，则可能需要安装其他依赖项。运行 Rasa NLU 时，它会检查是否安装了所有需要的依赖关系，并告诉你哪些缺失，如果有的话。

注意

如果你想确保你已经安装了所有组件可能需要的依赖项，并且你不介意附加的依赖关系，你可以使用

```
pip install -r alt_requirements/requirements_full.txt
```

而不是 `requirements.txt` 安装所有要求。

设置后端

您可以与 Rasa NLU 一起使用的大多数处理管道都需要安装 MITIE，spaCy 或 sklearn。

最适合大多数人：spaCy + sklearn

您也可以结合使用这两种运行方式。

安装 spacy 只是需要（欲了解更多信息，请访问 spacy 文件）：

```
pip install rasa_nlu[spacy]
```

```
python -m spacy download en_core_web_md
python -m spacy link en_core_web_md en
```

我们强烈建议至少使用“中等”大小的模型（`_md`），而不要使用 spacy 的默认小 `en_core_web_sm` 模型。小型车型也会起作用，不利之处在于它们在意图分类时性能较差。

注意

使用 spaCy 作为 Rasa 的后端是**首选选项**。对于大多数领域而言，性能与 MITIE 取得的成果相比，性能更好或同等。此外，设置更容易，训练速度更快。

第一选择：MITIE

该 MITIE 后端是包罗万象的，因为它同时提供了 NLP 和 ML 零件感。

```
pip install git+https://github.com/mit-nlp/MITIE.git
pip install rasa_nlu[mitie]
```

然后下载 MITIE 模型。你需要的文件是 `total_word_feature_extractor.dat`。将它保存到某个地方并 `config.json` 添加到其中 。

```
'mitie_file' : '/path/to/total_word_feature_extractor.dat'
```

警告

训练 MITIE 在数据集上的速度可能会比较慢，但意图较少。你可以试试

- 改为使用 sklearn + MITIE 后端（使用 sklearn 进行训练）或者
- 你可以安装我们的控制叉，这也应该减少训练时间。

另一个选择：sklearn + MITIE

还有第三个后端结合了前两个的优点：

1. sklearn 快速和良好的意图分类
2. 来自 MITIE 的良好实体识别和特征向量创建

尤其是，如果您有更多的意图（超过 10 个），使用 MITIE 的训练意向分类器可能需要很长时间。

要使用此后端，您需要按照说明安装 sklearn 和 MITIE。

教程：一个简单的餐厅搜索机器人

注意

请参阅迁移现有的应用程序以了解如何克隆现有的 wit / LUIS / Dialogflow 应用程序。

作为一个例子，我们将开始一个涵盖搜索餐馆领域的新项目。我们将从这些对话的一个非常简单的模型开始。你可以从那里建立起来。

让我们假设我们的机器人用户所说的*任何东西*都可以归类为以下一种**意图**：

- `greet`
- `restaurant_search`
- `thankyou`

当然，我们的用户可能有很多方式可以使用 `greet` 我们的 bot：

- *Hi!*
- *Hey there!*
- *Hello again :)*

甚至有更多的方式说你想找餐馆：

- *Do you know any good pizza places?*
- *I'm in the North of town and I want chinese food*
- *I'm hungry*

拉萨的第一份工作是 NLU 任何给定的句子分配到的一个**意图**类：`greet`，`restaurant_search`，或 `thankyou`。

第二个任务是标记的话就像"墨"和"中心"作为 `cuisine` 和 `location` **实体**分别。在本教程中，我们将构建一个完全符合这一点的模型。

准备训练数据

训练数据对开发聊天机器人非常重要。它应该包括要解释的文本以及我们期望 chatbots 将文本转换为文本的结构化数据（意图/实体）。获得训练文本的最佳方式是来自*真实用户*，获取结构化数据的最佳方式是自己假装自己是机器人。但为了帮助您入门，我们保存了一些数据。

下载该文件（json 格式）并将其打开，您将看到一个训练示例列表，其中每个示例均由以下组成`"text"`，`"intent"`并且`"entities"`如下所示。在你的工作目录中，创建一个 `data` 文件夹，并 `demo-rasa.json` 在那里复制这个文件。

```
{
  "text": "hey",
  "intent": "greet",
  "entities": []
```

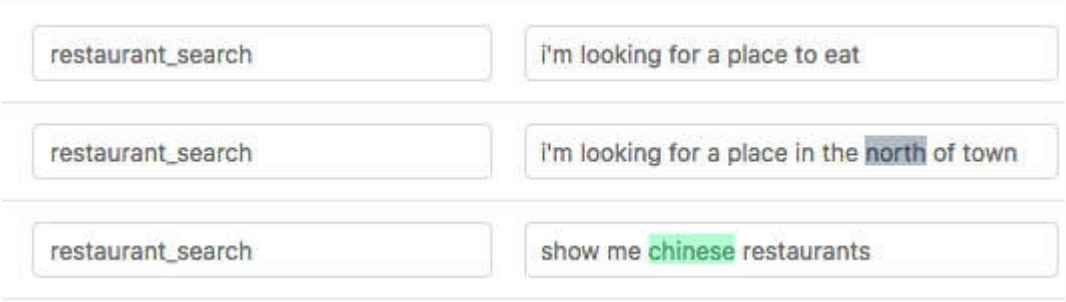
```
}
{
  "text": "show me chinese restaurants",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 8,
      "end": 15,
      "value": "chinese",
      "entity": "cuisine"
    }
  ]
}
```

希望这个格式很直观，如果你已经阅读了本教程，详细信息请参阅训练数据格式。否则，下一节“可视化训练数据”可帮助您更好地阅读，验证和/或修改训练数据。

可视化训练数据

它总是一个好主意，*看看*你的数据之前，期间和训练模式后。幸运的是，有一个用 rasa 格式创建训练数据的好工具。- 由@azazdeaz 创建- 对于检查和修改现有数据也非常有帮助。

对于演示数据，输出应该如下所示：



这是**强烈**建议您在 GUI 训练之前，要查看你的训练数据。

为您的项目训练新模型

现在我们要创建一个配置文件。确保首先设置了后端，请参阅安装。创建一个名为 `config_spacy.json` or 的文件 `config_mitie.json`，取决于所选的管道，在你的工作目录中看起来像这样

```
{
  "pipeline": "spacy_sklearn",
  "path" : "./projects",
  "data" : "./data/examples/rasa/demo-rasa.json"
}
```

或者你已经安装了 MITIE 后端：

```
{
  "pipeline": "mitie",
  "mitie_file": "./data/total_word_feature_extractor.dat",
  "path" : "./projects",
  "data" : "./data/examples/rasa/demo-rasa.json"
}
```

现在我们可以通过运行以下训练一个空间模型：

```
$ python -m rasa_nlu.train -c sample_configs/config_spacy.json
```

如果您想了解更多关于参数的信息，请参阅配置概述。几分钟后，Rasa NLU 将完成训练，并且您将 `projects/default/model_YYYYMMDD-HHMMSS` 在训练结束时看到一个新文件夹，其名称 与时间戳一样。

使用你的模型

默认情况下，服务器将查找 `path` 配置中指定的目录下的所有项目文件夹。当没有指定项目时，如本例所示，将使用“默认”项目，它本身使用最新的训练模型。

```
$ python -m rasa_nlu.server -c sample_configs/config_spacy.json
```

有关启动服务器的更多信息，请参阅使用 Rasa NLU 作为 HTTP 服务器。

然后您可以通过发送请求来测试新模型。在您的终端上打开一个新的选项卡/窗口并运行

注意

对于 Windows 用户，Windows 命令行界面不喜欢单引号。在必要时使用双引号并转义。

```
$ curl -X POST localhost:5000/parse -d '{"q":"I am looking for Mexican food"}' | python -m json.tool
```

应该返回

```
{
    "intent": {
    "name": "restaurant_search",
```

```
    "confidence": 0.8231117999072759
    },
    "entities": [
        {
            "start": 17,
            "end": 24,
            "value": "mexican",
            "entity": "cuisine",
            "extractor": "ner_crf"
        }
    ],
    "intent_ranking": [
        {
            "name": "restaurant_search",
            "confidence": 0.8231117999072759
        },
        {
            "name": "affirm",
            "confidence": 0.07618757211779097
        },
        {
            "name": "goodbye",
            "confidence": 0.06298664363805719
        },
        {
            "name": "greet",
            "confidence": 0.03771398433687609
        }
    ],
    "text": "I am looking for Mexican food"
}
```


如果您使用的是 `spacy_sklearn` 后端，并且找不到实体，请不要惊慌！本教程只是一个玩具示例，只有很少的训练数据才能期望获得良好的性能。

注意

意图分类与实体提取无关，例如在“我正在寻找中国食物”中，实体不被提取，尽管意图分类是正确的。

Rasa NLU 还将打印 `confidence` 意图分类的值。对于使用空间意图分类的模型，这将是一个概率。对于 MITIE 模型，这只是一个分数，**可能大于 1**。

你可以使用它在你的聊天机器人中进行一些错误处理（例如：再次询问用户是否信心不足），这对确定哪些意图需要更多的训练数据也很有帮助。

注意

输出可能包含其他或更少的属性，具体取决于您正在使用的管道。例如，`mitie` 管道不包含`"intent_ranking"`（参见下面的示例），而 `spacy_sklearn` 管道则包含（参见上面的示例）。

由于数据非常少，Rasa NLU 在某些情况下可以概括概念，例如：

```
$ curl -X POST localhost:5000/parse -d '{"q":"I want some italian food"}' | python -m json.tool
{
    "intent": {
        "name": "restaurant_search",
        "confidence": 0.5792111723774511
    },
    "entities": [
        {
            "entity": "cuisine",
            "value": "italian",
            "start": 12,
            "end": 19,
            "extractor": "ner_mitie"
        }
    ],
    "text": "I want some italian food"
}
```

即使在用来训练模型的例子中没有什么比这句话更像。要构建更强大的应用程序，您显然希望使用更多的训练数据，因此请去收集它！

配置

您可以通过以下方式向 Rasa NLU 提供选项：

- 一个 json 格式的配置文件
- 环境变量
- 命令行参数

环境变量覆盖配置文件中的选项，命令行参数将覆盖其他地方指定的任何选项。环境变量是大写字母，并带有前缀 `RASA_`，因此该选项 `pipeline` 是使用 `RASA_PIPELINE` env var 指定的。

默认

这是包含所有可用参数的默认配置：

```
{
  "project": null,
  "fixed_model_name": null,
  "pipeline": [],
  "language": "en",
  "num_threads": 1,
  "max_training_processes": 1,
  "path": "projects",
  "response_log": "logs",
  "storage": null,
  "config": "config.json",
```

```
  "log_level": "INFO",
  "port": 5000,
  "data": null,
  "emulate": null,
  "log_file": null,
  "mitie_file": "data/total_word_feature_extractor.dat",
  "spacy_model_name": null,
  "token": null,
  "cors_origins": [],
  "aws_endpoint_url": null,
  "max_number_of_ngrams": 7,
  "duckling_dimensions": null,
  "duckling_http_url": null,

  "ner_crf": {
    "BILOU_flag": true,
    "features": [
      ["low", "title", "upper", "pos", "pos2"],
      ["bias", "low", "word3", "word2", "upper", "title", "digit", "pos", "pos2",
"pattern"],
      ["low", "title", "upper", "pos", "pos2"]],
    "max_iterations": 50,
    "L1_c": 1,
    "L2_c": 1e-3
  },

  "intent_classifier_sklearn": {
    "C": [1, 2, 5, 10, 20, 100],
    "kernel": "linear"
  }
}
```

选项

每个配置值的简短说明和示例。

项目

类型： `str`

例子： `"my_project_name"`

描述： 定义项目名称以训练新模型并在使用 http 服务器时参考。默认值是 `null`

将导致使用默认项目`"default"`。所有项目都存储在`path`目录下。

管道

类型： `str` 要么 `[str]`

例子： `"mitie"` 要么 `["nlp_spacy", "ner_spacy", "ner_synonyms"]`

描述： 管道用于训练。可以是模板（传递字符串）或组件列表（数组）。有关所有可用模板，请参阅处理管道。

语言

类型： `str`

例子： `"en"` 要么 `"de"`

描述： 训练模型的语言。使用该语言将加载基础词汇向量

NUM_THREADS

类型： `int`

例子： `4`

描述： 训练期间使用的线程数（不是所有组件均支持，但其中一些可能仍然是单线程的 !）。

fixed_model_name

类型： `str`

例子： `"my_model_name"`

描述： 而不是生成模型名称（例如 `model_20170922-234435`）将使用固定的模型名称。该模型将始终保存在路径中 `{project_path}/{project_name}/{model_name}`。如果模型被分配了固定名称，则可能会覆盖之前训练过的模型。

max_training_processes

类型： `int`

例子： `1`

描述： 用于处理训练请求的进程数量。增加此值将对内存使用量产生很大影响。建议保持默认值。

路径

类型： `str`

例子： `"projects/"`

描述： 项目目录训练模型将被保存到（训练）并从（http 服务器）加载。

response_log

类型： `str` 要么 `null`

例子： `"logs/"`

描述： 日志将被保存的目录（包含查询和响应）。如果设置为 `null` 日志记录将被禁用。

配置

类型： `str`

例子： `"sample_configs/config_spacy.json"`

描述： 配置文件的位置（只能设置为 env var 或命令行选项）。

LOG_LEVEL

类型： `str`

例子： `"DEBUG"`

描述： 日志级别用于从框架内部输出消息。

港口

类型： `int`

例子： `5000`

描述： 在其上运行 http 服务器的端口。

数据

类型： `str`

例子： `"data/example.json"`

描述： 训练数据的位置。对于 JSON 和降价数据，这可以是单个文件或包含多个训练数据文件的目录。

cors_origins

类型： `list`

例子： `["*"]`，`["*.mydomain.com", "api.domain2.net"]`

描述： 允许 CORS(跨源资源共享)调用的域模式列表。默认值是`[]`禁止所有 CORS 请求。

仿真

类型： `str`

例子： `"wit"`，`"luis"`或者`"api"`

描述： 格式由 http 服务器返回。如果 `null`（默认）将使用 Rasa NLU 内部格式。否则，输出将根据指定的 API 格式化。

mitie_file

类型： `str`

例子： `"data/total_word_feature_extractor.dat"`

描述： 包含文件 `total_word_feature_extractor.dat`（请参阅安装）

spacy_model_name

类型： `str`

例子： `"en_core_web_md"`

描述： 如果要使用的模型 spacy 具有名称是从语言标签（不同`"en"`，`"de"`等），可使用此配置变量指定的型号名称。该名称将被传递给 `spacy.load(name)`。

代币

类型： `str` 要么 `null`

例子： `"asd2aw3r"`

描述： 如果设置，所有对服务器的请求都必须有`?token=<token>`查询参数。请参阅授权

max_number_of_ngrams

类型： `int`

例子： `10`

描述： 使用字符 ngrams 扩充特征向量时要使用的最大 ngram 数（`intent_featurizer_ngrams`仅限组件）

duckling_dimensions

类型： `list`

例子： `["time", "number", "amount-of-money", "distance"]`

描述： 定义鸭子组件将提取哪些维度，即实体类型。duckling 文档中可以找到完整的可用尺寸列表。

存储

类型： `str`

例子： `"aws"` 要么 `"gcs"`

描述： 存储类型为 persistor。有关更多详细信息，请参阅 Model Persistence。

BUCKET_NAME

类型： `str`

例子： `"my_models"`

描述： 存储模型的云存储桶的名称。如果指定的存储桶名称不存在，rasa 将创建它。有关更多详细信息，请参阅 Model Persistence。

aws_region

类型： `str`

例子： `"us-east-1"`

描述： 要使用的 aws 区域的名称。这仅在`"storage"`选择时用于`"aws"`。有关更多详细信息，请参阅 Model Persistence。

aws_endpoint_url

类型： `str`

例子： `"http://10.0.0.1:9000"`

描述： 自定义 S3 兼容存储提供程序的可选终端。这仅在`"storage"`选择时用于`"aws"`。有关更多详细信息，请参阅 Model Persistence。

ner_crf

特征

类型： `[[str]]`

例子： `[["low", "title"], ["bias", "word3"], ["upper", "pos", "pos2"]]`

描述： 这些特征是一个数组，其中包含 before，word，之后保存每个单词使用哪些特征的键，例如，在数组之前将具有“是标题大小写中的前面的单词吗？”。提供的功能有：，，，，，，，和`[before, word, after]"title"lowtitleword3word2pospos2biasupperdigit`

BILOU_flag

类型： `bool`

例子： `true`

描述： 该标志决定是否使用 BILOU 标签。BILOU 标记更严格，但每个实体需要更多示例。经验法则：仅在每个实体超过 100 个示例时使用。

max_iterations

类型： `int`

例子： `50`

描述： 这是在训练之前赋予 sklearn_crfcuite.CRF 标记的价值。

L1_C

类型： `float`

例子： `1.0`

描述： 这是在训练之前赋予 sklearn_crfcuite.CRF 标记的价值。指定 L1 正则化系数。

L2_C

类型： `float`

例子： `1e-3`

描述： 这是在训练之前赋予 sklearn_crfcuite.CRF 标记的价值。指定 L2 正则化系数。

intent_classifier_sklearn

C

类型： `[float]`

例子： `[1, 2, 5, 10, 20, 100]`

描述： 指定要为 C-SVM 进行交叉验证的正则化值的列表。这与 `kernel`GridSearchCV 中的超参数一起使用。

核心

类型： `string`

例子： `"linear"`

描述： 指定用于 C-SVM 的内核。这与 `C`GridSearchCV 中的超参数一起使用。

迁移现有的应用程序

Rasa NLU 旨在尽可能简单地从 wit / LUIS / Dialogflow 进行迁移。TLDR 迁移指令是：

- 从 wit / LUIS / Dialogflow 下载您的应用程序数据导出
- 按照教程：一个简单的餐厅搜索机器人，使用您下载的数据，而不是 `demo-rasa.json`

香蕉皮

只需注意一些您可能想要迁移的服务的具体内容即可

wit.ai

机智习惯于 `intents` 本地处理。现在他们有点混淆了。要创造一个 `intent` 智慧，你必须创造和 `entity` 跨越整个文本。您需要从您的下载文件中调用该文件 `expressions.json`

LUIS.ai

没什么特别的。下载数据并将其导入到 Rasa NLU 应该没有问题

Dialogflow

Dialogflow 导出会生成多个文件，而不仅仅是一个。把他们全部放在一个目录中（见 `data/examples/dialogflow` 回购），并将该路径传递给训练师。

仿真

为了使 Rasa NLU 易于使用现有项目，服务器可以*模拟* wit，LUIS 或 Dialogflow。在纯模式下，请求/响应如下所示：

```
$ curl -XPOST localhost:5000/parse -d '{"q":"I am looking for Chinese food"}' | python
-mjson.tool
{
  "text": "I am looking for Chinese food",
  "intent": "restaurant_search",
  "confidence": 0.4794813722432127,
  "entities": [
    {
      "start": 17,
      "end": 24,
      "value": "chinese",
      "entity": "cuisine"
    }
  ]
}
```

如果我们运行在 `wit` 模式（例如）`python -m rasa_nlu.server -e wit`

那么必须做出 GET 请求

```
$ curl 'localhost:5000/parse?q=hello' | python -mjson.tool
[
    {
        "_text": "hello",
        "confidence": 0.4794813722432127,
        "entities": {},
        "intent": "greet"
    }
]
```

对于 LUIS 也是如此，但响应格式略有不同

```
$ curl 'localhost:5000/parse?q=hello' | python -mjson.tool
{
    "entities": [],
    "query": "hello",
    "topScoringIntent": {
        "intent": "inform",
        "score": 0.4794813722432127
    }
}
```

最后是 Dialogflow

```
$ curl 'localhost:5000/parse?q=hello' | python -mjson.tool
{
    "id": "ffd7ede3-b62f-11e6-b292-98fe944ee8c2",
    "result": {
        "action": null,
        "actionIncomplete": null,
        "contexts": [],
        "fulfillment": {},
        "metadata": {
            "intentId": "ffdbd6f3-b62f-11e6-8504-98fe944ee8c2",
            "intentName": "greet",
            "webhookUsed": "false"
        },
        "parameters": {},
        "resolvedQuery": "hello",
        "score": null,
        "source": "agent"
    },
    "sessionId": "ffdbd814-b62f-11e6-93b2-98fe944ee8c2",
    "status": {
        "code": 200,
        "errorType": "success"
    },
    "timestamp": "2016-11-29T12:33:15.369411"
}
```

训练数据格式

Rasa NLU 的训练数据分为不同的部分 `common_examples`， `entity_synonyms` 和 `regex_features`。最重要的是 `common_examples`。

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

将 `common_examples` 被用来训练同时在实体和意图模型。你应该把所有的训练样例放入 `common_examples` 数组中。下一节详细描述一个例子的样子。正则表达式功能是帮助分类器检测实体或意图并提高性能的工具。

您可以使用 Chatito，一种使用简单的 DSL 以 rasa 格式生成训练数据集的工具。

常见的例子

常见的例子有三个组成部分：`text`， `intent`，和 `entities`。前两个是字符串，而最后一个是数组。

- *文本*是查询对象; 是需要被解析的内容。[需要]
- *意图*是与文字有关的意向。[可选的]
- 该*实体*是需要被识别的文本的特定部分。[可选的]

实体用 `start` 和 `end` 值指定，它们一起使 python 样式范围适用于字符串，例如，在下面的示例中，使用，然后。实体可以跨越多个单词，实际上该字段不必完全对应于您的示例中的子字符串。这样，您可以将同义词或拼写错误映射到相同的位置。`text="show me chinese restaurants"text[8:15] == 'chinese'valuevalue`

```
{
  "text": "show me chinese restaurants",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 8,
```

```
      "end": 15,
      "value": "chinese",
      "entity": "cuisine"
    }
  ]
}
```

实体同义词

如果您将实体定义为具有相同的值，则它们将被视为同义词。这是一个例子：

```
[
  {
    "text": "in the center of NYC",
    "intent": "search",
    "entities": [
      {
        "start": 17,
        "end": 20,
        "value": "New York City",
        "entity": "city"
      }
    ]
  },
  {
    "text": "in the centre of New York City",
    "intent": "search",
    "entities": [
      {
        "start": 17,
        "end": 30,
        "value": "New York City",
        "entity": "city"
      }
    ]
  }
]
```

正如你所看到的，即使第一个例子中的文本说明，实体在这两个例子中都 `city` 具有值。通过将 value 属性定义为与实体的开始和结束索引之间的文本中找到的值不同，您可以定义同义词。只要找到相同的文本，该值将使用同义词而不是消息中的实际文本。`New York CityNYC`

要使用训练数据中定义的同义词，您需要确保管道包含 `ner_synonyms` 组件（请参阅处理管道）。

或者，您可以添加一个“entity_synonyms”数组来为一个实体值定义多个同义词。这是一个例子：

```
{
  "rasa_nlu_data": {
    "entity_synonyms": [
      {
        "value": "New York City",
        "synonyms": ["NYC", "nyc", "the big apple"]
      }
    ]
  }
}
```

注意

请注意，使用上述格式添加同义词不会改进模型对这些实体的分类。 **实体在被替换为同义词值之前必须进行适当的分类。**

正则表达式特征

正则表达式可用于支持意图分类和实体提取。例如，如果您的实体具有某个邮政编码的特定结构，则可以使用正则表达式来简化该实体的检测。对于邮政编码示例，它可能如下所示：

```
{
    "rasa_nlu_data": {
        "regex_features": [
            {
                "name": "zipcode",
                "pattern": "[0-9]{5}"
            },
            {
                "name": "greet",
                "pattern": "hey[^\s]*"
            },
        ]
    }
}
```

这个名字并没有定义实体和意图，它只是一个人类可读的描述，让你记住这个正则表达式用于什么。正如您在上面的例子中看到的，您还可以使用正则表达式功能来提高意图分类性能。

尝试以尽可能少的词匹配的方式创建正则表达式。例如使用 `hey[^\s]*` 而不是 `hey.*`，因为后面的一个可能会匹配整个消息，而第一个只匹配一个单词。

用于实体提取的正则表达式功能目前仅由 `ner_crf` 组件支持！因此，其他实体提取器 `ner_mitie` 将不会使用生成的特征，并且它们的存在不会改进这些提取器的实体识别。目前，所有意图分类器都使用可用的正则表达式功能。

注意

正则表达式的功能不定义实体或意图！他们只是提供模式来帮助分类者识别实体和相关意图。因此，您仍然需要提供意图和实体示例作为您的训练数据的一部分！

Markdown 格式

训练数据可以使用以下标记格式（尚未支持 Regex 功能）。示例使用无序列表语法列出，例如减号 `-` 或星号 `*`：

```
## intent:check_balance
- what is my balance <!-- no entity -->
- how much do I have on my [savings](source_account) <!-- entity "source_account" has value "savings" -->
- how much do I have on my [my savings account](source_account:savings) <!-- synonyms, method 1-->

## intent:greet
- hey
- hello

## synonym:savings   <!-- synonyms, method 2 -->
- pink pig
```

组织

训练数据可以存储在单个文件中，也可以分成多个文件。对于较大的训练样例，将训练数据分成多个文件（例如每个意向一个文件）在维护性方面可能是有益的。

目前不支持在一个目录中存储具有不同文件格式（例如降价和 JSON）的文件。

注意

将训练数据分割成多个文件，目前只有 Markdown 和 JSON 数据。对于其他文件格式，您必须使用单文件方式。

使用 Rasa NLU 作为 HTTP 服务器

注意

在你使用服务器之前，你需要训练一个模型！请参阅为项目训练新模型

HTTP api 的存在使非 python 项目可以轻松使用 Rasa NLU，并且使当前使用 wit / LUIS / Dialogflow 的项目可以轻松尝试。

运行服务器

您可以运行一个简单的 http 服务器，该服务器使用您的项目处理请求：

```
$ python -m rasa_nlu.server -c sample_configs/config_spacy.json
```

服务器将 `path` 在配置中的参数定义的文件夹下查找现有项目。默认情况下，一个项目将加载最新的训练模型。

仿真

Rasa NLU 可以通过使`/parse`端点与现有代码兼容来“模拟”这三种服务中的任何一种。要激活它，可以添加到你的配置文件或运行服务器。例如，如果您通常会将您的文本发送给 LUIS 进行解析，那么您可以提出请求

`'emulate' : 'luis'-e luisGET`

`https://api.projectoxford.ai/luis/v2.0/apps/<app-id>?q=hello%20there`

在 luis 模拟模式下，您只需发送此请求即可呼叫 rasa

`http://localhost:5000/parse?q=hello%20there`

rasa 会忽略任何额外的查询参数，因此您可以安全地将它们一起发送。

端点

`POST /parse` （没有模拟）

你必须以这种格式发布数据，你可以这样做`'{"q":"<your text to parse>"}'`

```
$ curl -XPOST localhost:5000/parse -d '{"q":"hello there"}'
```

默认情况下，当查询中未指定项目时，`"default"`将使用该项目。您可以（应该）指定您要在查询中使用的项目：

```
$ curl -XPOST localhost:5000/parse -d '{"q":"hello there", "project":
"my_restaurant_search_bot"}
```

默认情况下，该项目的最新训练模型将被加载。您还可以针对项目的特定模型进行查询：

```
$ curl -XPOST localhost:5000/parse -d '{"q":"hello there", "project":
"my_restaurant_search_bot", "model": <model_XXXXXX>}
```

POST /train

您可以将您的训练数据发布到此端点，以训练项目的新模型。此请求将等待服务器答案：模型成功训练或训练出错。使用 HTTP 服务器，您必须指定您想要训练新模型的项目，以便稍后在解析请求时使用它：`/train?project=my_project`。通过查询字符串传递的任何参数都将被视为模型的配置参数，因此您可以通过传递其名称和调整后的值来更改配置部分中列出的所有配置值。

```
$ curl -XPOST localhost:5000/train?project=my_project -d
@data/examples/rasa/demo-rasa.json
```

您无法为已经训练新模型的项目发送训练请求（请参阅下文）。

GET /status

这将返回所有当前可用的项目，它们的状态（`training`或`ready`）以及它们的模型加载到内存中。还会返回服务器可用来完成`/parse`请求的可用项目列表。

```
$ curl localhost:5000/status | python -mjson.tool

{
  "available_projects": {
    "my_restaurant_search_bot" : {
      "status" : "ready",
      "available_models" : [
        <model_XXXXXX>,
        <model_XXXXXX>
      ]
    }
  }
}
```

GET /version

这将返回当前版本的 Rasa NLU 实例。

```
$ curl localhost:5000/version | python -mjson.tool
{
  "version" : "0.8.2"
}
```

`GET /config`

这将返回当前正在运行的 Rasa NLU 实例的配置。

```
$ curl localhost:5000/config | python -mjson.tool
{
    "config": "/app/rasa_shared/config_mitie.json",
    "data": "/app/rasa_nlu/data/examples/rasa/demo-rasa.json",
    "duckling_dimensions": null,
    "emulate": null,
    ...
  }
```

授权

为了保护您的服务器，您可以在 Rasa NLU 配置中指定一个令牌，例如通过添加到您的配置文件或设置环境变量。如果设置，则该标记必须在所有请求中作为查询参数传递，例如：`"token" : "12345"RASA_TOKEN`

```
$ curl localhost:5000/status?token=12345
```

在默认的 CORS（跨源资源共享）调用是不允许的。如果您想从另一个域（例如来自训练 Web UI）调用您的 Rasa NLU 服务器，则可以将该域添加到配置值中，将该域添加到白名单中 `cors_origin`。

提供多个应用程序

根据您选择的后端，Rasa NLU 可以使用相当多的内存。因此，如果您在生产中使用多个模型，您希望从相同的过程中提供这些模型并避免重复内存负载。

尽管这会节省后端加载相同的后端两次，但仍需要加载一组后端
每个语言和后端的单词向量（构成大部分内存消耗）。

如前所述，Rasa NLU 自然可以处理多个应用程序：默认情况下，服务器将加载 `path` 在配置中定义的目录下找到的所有项目。下面的文件结构如下：

`path directory`

- <path>
    - <project_A>

    - <model_XXXXXX>
    - <model_XXXXXX>

...

- <project_B>
    - <model_XXXXXX>

...
...
所以你可以指定在你的`/parse`请求中使用哪一个：

```
$ curl 'localhost:5000/parse?q=hello&project=my_restaurant_search_bot'
```

要么

```
$ curl -XPOST localhost:5000/parse -d '{"q":"I am looking for Chinese food",
"project":"my_restaurant_search_bot"}'
```

您还可以指定您想要用于给定项目的模型，默认使用的是最新的训练：

```
$ curl -XPOST localhost:5000/parse -d '{"q":"I am looking for Chinese food",
"project":"my_restaurant_search_bot", "model":<model_XXXXXX>}'
```

如果该`path`目录下的服务器找不到任何项目，`"default"`则将使用一个使用简单回退模型的项目。

从 python 使用 Rasa NLU

除了将 Rasa NLU 作为 HTTP 服务器运行外，您还可以直接在您的 python 程序中使用它。Rasa NLU 同时支持 Python 2 和 Python 3。

训练时间

要创建模型，您可以按照与非 python 用户相同的说明进行操作。或者，您可以使用类似下面的脚本（使用 spacy）直接在 Python 中训练：

```
from rasa_nlu.converters import load_data
from rasa_nlu.config import RasaNLUConfig
from rasa_nlu.model import Trainer

training_data = load_data('data/examples/rasa/demo-rasa.json')
trainer = Trainer(RasaNLUConfig("sample_configs/config_spacy.json"))
trainer.train(training_data)
model_directory = trainer.persist('./projects/default/')  # Returns the directory the
model is stored in
```

预测时间

你可以直接从你的 python 脚本调用 Rasa NLU。为此，您需要加载模型的元数据并实例化一个解释器。该 `metadata.json` 模型中的目录包含了必要的信息，以恢复模式：

```
from rasa_nlu.model import Metadata, Interpreter

# where `model_directory points to the folder the model is persisted in
interpreter = Interpreter.load(model_directory,
RasaNLUConfig("sample_configs/config_spacy.json"))
```

然后您可以使用加载的解释器来解析文本：

```
interpreter.parse(u"The text I want to understand")
```

它返回与 `dict`HTTP API 相同的结果（没有仿真）。

如果创建了多个模型，则在不同模型之间共享组件是合理的。例如`'nlp_spacy'`，想要访问空间字向量的每个管道都使用的组件可以被缓存以避免在主存储器中多次存储大的字向量。要使用缓存，`ComponentBuilder`应在加载和训练模型时传递 a 。

这里有一个关于如何创建一个组件生成器的简短例子，它可以被重用来训练和运行多个模型，以训练一个模型：

```
from rasa_nlu.converters import load_data
from rasa_nlu.config import RasaNLUConfig
from rasa_nlu.components import ComponentBuilder
from rasa_nlu.model import Trainer

builder = ComponentBuilder(use_cache=True)      # will cache components between
pipelines (where possible)

training_data = load_data('data/examples/rasa/demo-rasa.json')
trainer = Trainer(RasaNLUConfig("sample_configs/config_spacy.json"), builder)
trainer.train(training_data)
model_directory = trainer.persist('./projects/default/')  # Returns the directory the
model is stored in
```

同一个构建器可以用来加载模型（可以是完全不同的）。构建器只缓存可安全地在模型之间共享的组件。以下是加载模型时如何使用构建器的简短示例：

```
from rasa_nlu.model import Metadata, Interpreter
config = RasaNLUConfig("sample_configs/config_spacy.json")

# For simplicity we will load the same model twice, usually you would want to use the metadata of
# different models

interpreter = Interpreter.load(model_directory, config, builder)     # to use the builder, pass it as an arg when loading the model
# the clone will share resources with the first model, as long as the same builder is passed!
interpreter_clone = Interpreter.load(model_directory, config, builder)
```

实体提取

有许多不同的实体提取组件，这对新用户来说似乎很吓人。在这里，我们将通过一些使用案例并提出使用建议。

零件	需要	模型	笔记
`ner_mitie`	MITIE	结构化 SVM	适合训练定制实体
`ner_crf`	crfsuite	条件随机场	适合训练定制实体
`ner_spacy`	spaCy	平均感知器	提供预先训练的实体
`ner_duckling`	duckling	上下文无关语法	提供预先训练的实体

确切需要的软件包可以在中找到 `dev-requirements.txt`，它们也应该在缺失时显示，并且使用需要它们的组件。

要改进实体提取，如果实体具有不同的格式（例如邮编），则可以使用正则表达式功能。更多信息可以在训练数据格式中找到。

注意

要使用这些组件，您可能需要定义自定义管道，请参阅处理管道。您可以将多个 ner 组件添加到您的管道中; 每个结果将在最终输出中结合起来。

用例

这里我们将概述实体提取的一些常见用例，并就使用哪些组件提出建议。

地点，日期，人员，组织

spaCy 在许多模型中都有出色的预先训练的命名实体识别器。你可以在这个令人敬畏的交互式演示中测试它们。我们不建议您尝试使用 spaCy 来训练您自己的 NER，除非您拥有大量数据并知道自己在做什么。请注意，一些 spaCy 模型非常区分大小写。

日期，金额，持续时间，距离，常规

该 duckling 包不转向，如“晚上 8 点下周四”表现为，您可以使用实际的日期时间对象的一个伟大的工作。它还可以处理诸如“两小时”，金钱数量，距离等的持续时间。幸运的是，还有一个用于 duckling 的 python 包装！您可以通过从 PyPI 安装 duckling 包并添加 `ner_duckling` 到您的管道来使用此组件。

自定义，特定于域的实体

在介绍性教程中，我们构建了一个餐厅机器人，并为位置和美食创建自定义实体。用于训练这些领域特定实体识别器的最佳组件是组件 `ner_mitie` 和 `ner_crf` 组件。建议您尝试使用这两种方法，以查看哪些数据集最适合您。

返回的实体对象

在解析后返回的对象中，有两个字段显示有关管道如何影响返回实体的信息。`extractor` 实体的字段告诉你哪个实体提取器找到了这个特定的实体。该 `processors` 字段包含更改此特定实体的组件的名称。

同义词的使用也可能导致 `value` 字段和 `text` 不完全匹配。相反，它会返回训练的同义词。

```
{
  "text": "show me chinese restaurants",
  "intent": "restaurant_search",
  "entities": [
```

```
    {
      "start": 8,
      "end": 15,
      "value": "chinese",
      "entity": "cuisine",
      "extractor": "ner_mitie",
      "processors": []
    }
  ]
}
```

从反馈中改进模型

当 rasa_nlu 服务器运行时，它会跟踪它所做的所有预测并将它们保存到日志文件中。默认情况下，日志文件被放入 `logs/`。该目录中的文件每行包含一个 json 对象。您可以修复任何不正确的预测并将其添加到您的训练集以改善解析器。在将这些数据添加到您的训练数据中之后，但在重新训练模型之前，强烈建议您使用可视化工具来发现任何错误，请参阅可视化训练数据。

模型持久化

Rasa NLU 支持使用 S3 和 GCS 来保存模型。

- **亚马逊 S3 存储**

  使用 `boto3` 您可以安装的模块支持 S3 。 `pip install boto3`

  `storage` 选项设置为启动 Rasa NLU 服务器 `aws`。获取您的 S3 凭证并设置以下环境变量：

  - `AWS_SECRET_ACCESS_KEY`
  - `AWS_ACCESS_KEY_ID`
  - `AWS_REGION`
  - `BUCKET_NAME`

- **Google 云端存储**

  GCS 支持使用 `google-cloud-storage` 您可以安装的软件包

  `pip install google-cloud-storage`

  `storage` 选项设置为启动 Rasa NLU 服务器 `gcs`。

  在谷歌应用引擎和计算引擎上运行时，已经设置了认证凭证。要在本地或其他地方运行，请查看他们的客户端回购以了解设置身份验证的详细信息。它涉及从谷歌云控制台创建服务帐户密钥文件，并将 `GOOGLE_APPLICATION_CREDENTIALS` 环境变量设置为该密钥文件的路径。

如果没有名字`$BUCKET_NAME`rasa 的桶会创建它。模型在保存到云之前进行 gzip 压缩。

语言支持

目前 Rasa NLU 已经过测试，可以用于以下语言：

后端	支持的语言
spacy-sklearn	英语（`en`），德语（`de`），西班牙语（`es`），葡萄牙语（`pt`），意大利语（`it`），荷兰语（`nl`），法语（`fr`）
MITIE	english（`en`）

这些语言可以设置为配置的一部分。

添加一种新语言

我们希望尽可能简化添加新语言的过程，以增加支持的语言数量。尽管如此，要使用某种语言，您需要经过训练的单词表示形式，或者需要使用该语言的大量文本数据来自行训练该演示文稿。

这些是添加新语言所需的步骤：

spacy-sklearn

spaCy 已经提供了关于添加语言的非常好的文档页面。这将帮助您在 spaCy 中训练一种新语言的标记器和词汇。

如文档中所述，您需要注册您的语言，使用 `set_lang_class()` 它将允许 Rasa NLU 加载并使用您的新语言，方法是将语言标识符作为 `language` 配置选项进行传递。

MITIE

1. 获取∽干净的语言语料库（维基百科转储工程）作为一组文本文件
2. 在您的语料库上构建并运行 MITIE wordrep 工具。这可能需要几个小时/天，具体取决于您的数据集和工作站。你需要 128GB 的 RAM 来让 wordrep 运行 - 是的，这很有用：尝试扩展你的交换。
3. 将新的路径设置 `total_word_feature_extractor.dat` 为 *mitie_file* 参数的值 `config_mitie.json`

处理管道

传入消息的过程分为不同的组件。这些组件在一个所谓的处理管道中一个接一个地执行。有实体提取的组件，用于意图分类，预处理，未来还会有更多。

每个组件处理输入并创建一个输出。输出可以由管道中的该组件后面的任何组件使用。有些组件只生成管道中其他组件使用的信息，还有其他组件会生成 `Output` 在处理完成后返回的属性。例如，对于输出的句子`"I am looking for Chinese food"`

```
{
    "text": "I am looking for Chinese food",
    "entities": [
        {"start": 8, "end": 15, "value": "chinese", "entity": "cuisine", "extractor":
"ner_crf"}
    ],
    "intent": {"confidence": 0.6485910906220309, "name": "restaurant_search"},
    "intent_ranking": [
        {"confidence": 0.6485910906220309, "name": "restaurant_search"},
        {"confidence": 0.1416153159565678, "name": "affirm"}
    ]
}
```

被创建为预配置管道中不同组件的结果的组合 `spacy_sklearn`。例如，该 `entities` 属性由 `ner_crf` 组件创建。

预先配置的管道

为了减轻提出自己的处理流水线的负担，我们提供了几个准备使用的模板，可以通过将 `pipeline` 配置值设置为您要使用的模板名称来使用它。以下是现有模板的列表：

模板名称	相应的管道
spacy_sklearn	`["nlp_spacy", "tokenizer_spacy", "intent_entity_featurizer_regex", "intent_featurizer_spacy", "ner_crf", "ner_synonyms", "intent_classifier_sklearn"]`
MITIE	`["nlp_mitie", "tokenizer_mitie", "ner_mitie", "ner_synonyms", "intent_entity_featurizer_regex", "intent_classifier_mitie"]`
mitie_sklearn	`["nlp_mitie", "tokenizer_mitie", "ner_mitie", "ner_synonyms", "intent_entity_featurizer_regex", "intent_featurizer_mitie", "intent_classifier_sklearn"]`
keyword	`["intent_classifier_keyword"]`

通过直接将组件的名称传递给 `pipeline` 配置变量中的 Rasa NLU 来创建自己的管道，例如。这创建了一个只能进行实体识别的管道，但没有意图分类。因此，

输出不会包含任何有用的意图。

```
"pipeline": ["nlp_spacy", "ner_crf", "ner_synonyms"]
```

内置组件

每个组件及其属性的简短说明。如果你正在寻找更多的细节，你应该看看组件的相应源代码。`Output`描述了每个组件添加到处理消息的最终输出结果的内容。如果没有输出，该组件很可能是另一个组件的预处理器。

nlp_mitie

短： MITIE 初始化程序

输出： 没有

描述： 初始化控制结构。每个细菌组分都依赖于此，因此应该将其放置在使用任何细菌组分的每条管道的开始处。

nlp_spacy

短： 空间语言初始化器

输出： 没有

描述： 初始化空间结构。每个 Spacy 组件都依赖于这个，因此这应该放在使用任何 Spacy 组件的每个管道的开始处。

intent_featurizer_mitie

短： MITIE 意图特征

输出： 什么都不做，用作需要意向特征的意图分类器的输入（例如`intent_classifier_sklearn`）

使用 MITIE 特征创建用于意图分类的特征。

描述： 注意

不被`intent_classifier_mitie`组件使用。目前，只能`intent_classifier_sklearn`使用预先计算的功能。

intent_featurizer_spacy

短： 空间意图 featurizer

输出： 什么都不做，用作需要意向特征的意图分类器的输入（例如 `intent_classifier_sklearn`）

描述： 使用 spacy featurizer 创建用于意图分类的特征。

intent_featurizer_ngrams

短： 将 char-ngram 特征追加到特征向量中

输出： 什么也没有，将其特征附加到由另一个意图特征生成的现有特征向量

描述： 这个特征向特征向量附加字符 ngram 特征。在训练期间，组件查找最常见的字符序列（例如 `app` 或 `ing`）。如果字符序列存在于单词序列中，则添加的功能表示布尔标志。

意

在此之前需要有另一个意向特征！

intent_classifier_keyword

短： 简单的关键字匹配意图分类器。

输出： `intent`

输出实施例：

```
{
    "intent": {"name": "greet", "confidence": 0.98343}
}
```

描述： 这个分类器主要用作占位符。它能够通过在传递的消息中搜索这些关键字来识别*问候语*和 *再见*意图。

intent_classifier_mitie

短： MITIE 意向分类器（使用文本分类器）

输出： `intent`

输出实施例：

```
{
    "intent": {"name": "greet", "confidence": 0.98343}
}
```

描述： 这个分类器使用 MITIE 来执行意图分类。底层分类器使用具有稀疏线性内核的多类线性 SVM（请参阅 mitie 训练员代码）。

intent_classifier_sklearn

短： sklearn 意图分类器

输出： `intent` 和 `intent_ranking`

输出实施例：

```
{
    "intent": {"name": "greet", "confidence": 0.78343},
    "intent_ranking": [
        {
            "confidence": 0.1485910906220309,
            "name": "goodbye"
        },
        {
            "confidence": 0.08161531595656784,
            "name": "restaurant_search"
        }
    ]
}
```

描述： Sklearn 意图分类器训练线性 SVM，该线性 SVM 使用网格搜索进行优化。除了其他分类器之外，它还提供没有“赢”的标签的排名。空间意图分类器需要在管道中加入特色功能。该特色功能创建用于分类的功能。

intent_entity_featurizer_regex

短： 创建正则表达式来支持意图和实体分类

输出： `text_features` 和 `tokens.pattern`

描述： 在训练过程中，正则表达式意图特征创建了一个在训练数据格式中定义的*正则表达式*列表。如果在输入中找到表达式，将设置一个特征，稍后将输入到意图分类器/实体提取器中以简化分类(假定分类器在训练阶段已经学习到该特征指示某个意图)。用于实体提取的正则表达式功能目前仅由 `ner_crf` 组件支持 ！

tokenizer_whitespace

短： 使用空格作为分隔符的 Tokenizer

输出： 没有

描述： 为每个以空格分隔的字符序列创建一个标记。可以用来为 MITIE 实体提取器定义 tokesn。

tokenizer_mitie

短： 使用 MITIE 的 Tokenizer

输出： 没有

描述： 使用 MITIE 标记器创建标记。可以用来定义 MITIE 实体提取器的标记。

tokenizer_spacy

短： 使用 spacy 的 Tokenizer

输出： 没有

描述： 使用空间标记器创建标记。可以用来定义 MITIE 实体提取器的标记。

ner_mitie

短： MITIE 实体提取（使用 mitie ner 训练师）

输出： 追加 `entities`

输出实施例：

```
{
    "entities": [{"value": "New York City",
                  "start": 20,
                  "end": 33,
                  "entity": "city",
                  "extractor": "ner_mitie"}]
}
```

描述： 这使用 MITIE entitiy 提取来查找消息中的实体。基础分类器使用具有稀疏线性内核和自定义特征的多类线性 SVM。

ner_spacy

短： 空间实体提取

输出： 追加 `entities`

输出实施例：

```
{
    "entities": [{"value": "New York City",
                  "start": 20,
                  "end": 33,
                  "entity": "city",
                  "extractor": "ner_spacy"}]
}
```

描述： 使用 spacy 这个组件可以预测消息的实体。spacy 使用统计 BILUO 转换模型。到目前为止，该组件只能使用空间内置的实体提取模型，并且不能再训练。

ner_synonyms

短： 将同义实体值映射到相同的值。

输出： 修改先前实体提取组件找到的现有实体

描述： 如果训练数据包含定义的同义词（通过使用 `value` 实体示例上的属性）。该组件将确保检测到的实体值将映射到相同的值。例如，如果您的训练数据包含以下示例：

```
[{
  "text": "I moved to New York City",
  "intent": "inform_relocation",
  "entities": [{"value": "nyc",
                "start": 11,
                "end": 24,
                "entity": "city",
               }]
},
{
  "text": "I got a new flat in NYC.",
  "intent": "inform_relocation",
  "entities": [{"value": "nyc",
                "start": 20,
                "end": 23,
```

```
                "entity": "city",
               }]
}]
```

该组件将允许你在实体映射和到。即使消息包含，实体提取也会返回。当这个组件改变一个存在的实体时，它将自己附加到这个实体的处理器列表中。`New York CityNYCnycnycNYC`

ner_crf

短： 条件随机场实体提取

输出： 追加 `entities`

输出实施例：

```
{
    "entities": [{"value":"New York City",
                  "start": 20,
                  "end": 33,
                  "entity": "city",
                  "extractor": "ner_crf"}]
}
```

描述： 该组件实现条件随机字段来执行命名实体识别。CRF 可以被认为是一个无向马尔可夫链，其时间步是词，状态是实体类。单词（大写，POS 标签等）的特征给出了某些实体类的概率，以及相邻实体标签之间的过渡：然后计算并返回最可能的一组标签。

ner_duckling

短： 将 duckling 支持添加到管道以统一实体类型（例如检索常见日期/数字格式）

输出： 追加 `entities`

输出实施例：

```
{
    "entities": [{"end": 53,
                  "entity": "time",
                  "start": 48,
                  "value": "2017-04-10T00:00:00.000+02:00",
                  "extractor": "ner_duckling"}]
}
```

描述： Duckling 允许识别日期，数字，距离和其他结构化实体并对

它们进行标准化（所有可用实体的参考请参阅 duckling 文档）。该组件识别由 duckling 尺寸配置变量定义的实体类型。请注意，duckling 试图在不提供排名的情况下提取尽可能多的实体类型。例如，如果您为鸭子组件指定了两者 `number` 并将其 `time` 作为维度，那么该组件将提取两个实体：`10` 作为数字和　作为文本的时间。在这种情况下，您的应用程序必须决定哪种实体类型是正确的。

```
in 10 minutesI will be there in 10 minutes
```

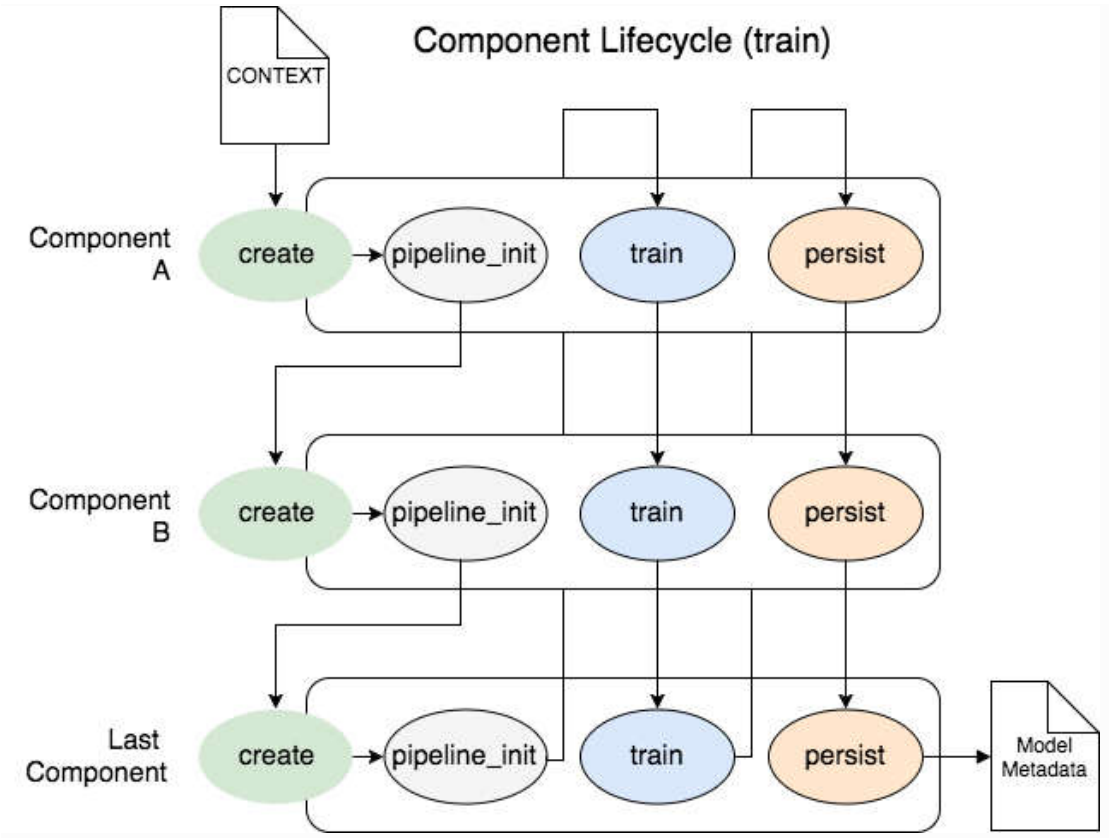
创建新的组件

目前，您需要依赖 Rasa NLU 随附的组件，但我们计划添加在您的代码中创建自己的组件的可能性。尽管如此，我们期待着您为新组件做出贡献（例如，进行情感分析的组件）。对代码的一瞥 `rasa_nlu.components.Component` 将揭示需要实施哪些功能来创建新组件。

组件生命周期

每个组件都可以从 `Component` 基类实现几种方法; 在一个管道中，这些不同的方法将以特定的顺序被调用。让我们假设，我们将下面的管道添加到我们的配置中：。该图显示了该管道训练期间的呼叫顺序：

```
"pipeline": ["Component A", "Component B", "Last Component"]
```



在使用 `create` 函数创建第一个组件之前，会创建一个所谓 `context` 的（它不过是一个 python 字典）。此上下文用于在组件之间传递信息。例如，一个组件可以计算训练数据的特征向量，将其存储在上下文中，另一个组件可以从上下文中检索这些特征向量并进行意图分类。

最初，上下文用所有配置值填充，图像中的箭头显示调用顺序并可视化传递的上下文的路径。在所有组件都被训练并保持之后，最终的上下文字典用于保存模型的元数据。

评估

评估脚本 *evaluate.py* 允许您测试您的模型性能以进行意图分类和实体识别。您可以调用此脚本提供测试数据，模型和配置文件参数：

```
$ python -m rasa_nlu.evaluate -d data/my_test.json -m models/my_model -c
my_nlu_config.json
```

如果您想使用交叉验证评估您的管道，则可以使用模式交叉验证标志运行评估脚本。这给你一个估计预测模型在实践中的准确程度。请注意，您无法在此模式下指定模型，因为将在每个交叉验证循环的部分数据上训练新模型。脚本的示例调用将是：

```
$ python -m rasa_nlu.evaluate -d data/examples/rasa/demo-rasa.json -c
sample_configs/config_spacy.json --mode crossvalidation
```

意向分类

评估脚本将记录每个意图的精确度，召回率和 f1 度量，并进行一次总结。此外，它会为您创建一个混淆矩阵，以查看哪些意图被误认为是其他人。

实体提取

对于每个实体提取器，评估脚本将记录其在实训数据中的每个实体类型的性能。因此，如果您在流水线中使用 `ner_crf` 并且 `ner_duckling` 在每个实体类型中记录两个包含召回率，精度和 f1 度量的评估表。

在 `ner_duckling` 我们实际上为每个定义的 duckling 尺寸运行评估的情况下。如果你使用 `time` 和 `ordinal` 维度，你会得到两个评估表：一个用于一个用于 。
`ner_duckling (Time)ner_duckling (Ordinal)`

`ner_synonyms` 不会创建评估表，因为它只会更改找到的实体的值，并且本身不会找到实体边界。

最后，请记住，测试数据中的实体类型必须与提取组件的输出相匹配。这一点尤其重要 `ner_duckling`，因为它不适合您的训练数据。

实体评分

为了评估实体提取，我们应用了一种简单的基于标签的方法 我们不考虑 BILOU 标签，而只考虑每个标记的实体类型标签。对于像“亚历山大广场附近”这样的地点实体，我们期望标签“LOC”“LOC”而不是基于 BILOU 的“B-LOC”“L-LOC”。我们的方法在评估时比较宽松，因为它会奖励部分提取，并且不会惩罚实体的分裂。例如，给定的上述实体“亚历山大广场附近”和提取“亚历山大广场”的系统，这将奖励“亚历山大广场”的提取并惩罚错过的“近”字。然而，基于 BILOU 的方法会将此标记为完全失败，因为它期望 Alexanderplatz 被标记为实体（L-LOC）中的最后一个令牌而不是单个令牌实体（U-LOC）。另请注意，

下面是“今晚亚历山大广场附近”这两个不同评分机制的比较：

提取	简单标签（分数）	BILOU 标签（分数）
[near Alexanderplatz](loc) [tonight](time)	loc loc time (3)	B-loc L-loc U-time (3)
[near](loc) [Alexanderplatz](loc) [tonight](time)	loc loc time (3)	U-loc U-loc U-time (1)
near [Alexanderplatz](loc) [tonight](time)	O loc time (2)	O U-loc U-time (1)
[near](loc) Alexanderplatz [tonight](time)	loc O time (2)	U-loc O U-time (1)

提取	简单标签（分数）	BILOU 标签（分数）
[near Alexanderplatz tonight](loc)	loc loc loc (2)	B-loc I-loc L-loc (1)

经常问的问题

我需要多少个训练样例？

不幸的是，对这个问题没有一丝不苟的答案。这取决于你的意图和你的实体。

如果您的意图很容易混淆，则需要更多的训练数据。因此，随着您添加更多意图，您还希望为每个意图添加更多的训练示例。如果你为每个意图快速写出 20-30 个独特的表达方式，那么你应该在开始的时候做好。

实体也是如此。您将需要的训练示例的数量取决于您的不同实体类型的密切相关程度以及您的用例中实体与非实体的区分程度。

要评估模型的性能，请运行服务器并手动测试一些消息，或使用评估脚本。

它与 Python 3 运行？

是的，Rasa NLU 支持 python 2.7 以及 python 3.5 和 3.6。如果特定 Python 版本有任何问题，请随时创建问题或直接提供修补程序。

支持哪些语言？

有包含所有 officialy 支持的语言列表在这里。不过，还有其他人正在努力增加更多的语言，请随时查看 github 问题 部分或 gitter 聊天。

我正在运行哪个版本的 Rasa NLU？

要找出你正在运行的 rasa 版本，你可以执行

```
$ python -c "import rasa_nlu; print(rasa_nlu.__version__);"
```

如果您使用虚拟环境来运行您的 python 代码，请确保您使用正确的 python 来执行上面的代码。

我为什么得到一个 `UndefinedMetricWarning`？

完整的警告是： 警告是缺乏训练数据的结果。在训练过程中，数据集将被分割多次，如果有任何意图的训练样本很少，分割可能会导致分割不包含此意图的任何示例。

```
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with
no predicted samples.
```

因此，解决方案是增加更多的训练样本。由于这只是一个警告，所以训练仍然会成功，但是由此产生的模型预测可能对您缺乏训练数据的意图较弱。

我有一个问题，你能帮助我吗？

我们很乐意帮助你。如果您不确定问题是否与您的设置有关，您应该在 gitter 聊天中说明您的问题。如果您发现该框架存在问题，请提交关于 github 问题的报告， 包括重现问题所需的所有信息。

迁移指南

此页面包含有关主要版本之间更改以及如何从一个版本迁移到另一个版本的信息。

0.9.x 至 0.10.0

- 我们引入了一个叫做 a 的新概念 `project`。您可以为项目训练多个版本的模型。例如，您可以训练初始模型并添加更多训练数据并重新训练该项目。这将导致同一项目的新模型版本。这使您可以从 http 服务器请求最新的型号版本，并使模型处理更加结构化。
- 如果要重新使用训练好的模型，则需要将它们移动到项目后命名的目录中。例如，如果您已在目录中获得训练有素的模型，则 `my_root/model_20170628-002704` 需要将其移至 `my_root/my_project/model_20170628-002704`。您的新项目名称将为 `my_project`，您可以使用 http 服务器使用查询模型 `curl http://localhost:5000/parse?q=hello%20there&project=my_project`
- 文档已移至 https://rasahq.github.io/rasa_nlu/
- 将 `name` 参数重命名为 `project`。这意味着您现在需要通过训练请求，例如包含训练数据的请求正文 `project parameter instead of ``namePOST /train?project=my_project_name`
- 调整远程云存储以支持项目。这是一个向后不兼容的更改，不幸的是，您需要重新训练上传的模型并重新上传它们。

0.8.x 到 0.9.x

- 添加 `tokenizer_spacy` 到训练过的 spacy_sklearn 模型元数据（在之后 `nlp_spacy`）。另一种方法是重新训练模型

0.7.x 至 0.8.x

- 对于空间实体提取的训练和加载能力已经下降，以支持新的 CRF 提取器。这意味着模型需要使用 crf 提取器进行再训练。
- 将参数和配置值名称 `backend` 更改为 `pipeline`。
- 模型元数据格式已经发生变化。您可以重新训练模型或更改存储的 metadata.json：
    - 重命名 `language_name` 为 `language`
    - 重命名 `backend` 为 `pipeline`
    - 为 MITIE 的模型，你需要更换 `feature_extractor` 有 `mitie_feature_extractor_fingerprint`。该指纹取决于您使用的语言，因为 `en` 它是。 `"mitie_feature_extractor_fingerprint": 10023965992282753551`

0.6.x 到 0.7.x

- 将参数和配置值名称 `server_model_dir` 更改为 `server_model_dirs`。
- 将参数和配置值名称 `write` 更改为 `response_log`。它现在配置日志应该写入的 *目录*（不是文件！）
- 模型元数据格式已更改。所有路径现在都相对于 `path` 训练和加载过程中指定的配置。如果你想运行与某个版本的分组训练 0.7 车型则需要在路径手动适应 `metadata.json` 从

```
{
    "trained_at": "20170304-191111",
    "intent_classifier": "model_XXXX_YYYY_ZZZZ/intent_classifier.pkl",
    "training_data": "model_XXXX_YYYY_ZZZZ/training_data.json",
    "language_name": "en",
    "entity_extractor": "model_XXXX_YYYY_ZZZZ/ner",
    "feature_extractor": null,
    "backend": "spacy_sklearn"
}
```

到沿着这条线（使所有路径相对于模型的基本目录，这是 `model_XXXX_YYYY_ZZZZ/`）：

```
{
    “trained_at” : “20170304-191111” ,
    “intent_classifier” : “intent_classifier.pkl” ,
    “training_data” : “training_data.json” ,
    “language_name” : “en” ,
    “entity_synonyms” : null ,
    “entity_extractor” : “ ner“ ,
    ”feature_extractor“ : null ,
    ”backend“ : ”spacy_sklearn“
}
```

许可证

Apache 许可证

版本 2.0，2004 年 1 月

http://www.apache.org/licenses/

使用，复制和分发的条款和条件

1.定义。

“许可证”是指使用，复制，
以及本文件第 1 至 9 节定义的分发。

“许可人”是指版权所有者或者授权的实体
授予许可的版权所有者。

“法人实体”是指代理实体和所有人的联合
其他控制，受控制或共同控制的实体
控制该实体。为了这个定义的目的，
“控制”是指（i）直接或间接的力量造成的
指导或管理此类实体，无论是通过合同还是通过
否则，或（ii）所有权百分之五十（50％）或以上的
已发行股份，或（iii）该实体的受益所有权。

“您”（或“您的”）是指个人或法人实体
行使本许可证授予的权限。

“来源”形式是指进行修改的首选形式，
包括但不限于软件源代码，文档
源和配置文件。

“对象”形式是指由机械产生的任何形式
转换或翻译源表单，包括但
不限于编译的目标代码，生成的文档，
并转换为其他媒体类型。

“作品”是指作者的作品，无论是来源还是作品
对象形式，根据许可证提供，如 a 所示
版权声明包含在作品中或作品中
（附录中提供了一个例子）。

“衍生作品”是指任何作品，无论是来源还是对象
形式，即基于（或源自）该作品并为之而生
编辑修订，注释，阐述或其他修改
作为一个整体，代表作者身份的原创作品。为了这个目的
衍生作品不得包含剩余的作品

可分离的，或仅仅链接（或按名称绑定）
作品及其衍生作品。

“贡献”是指任何作者的作品，包括作品
作品的原始版本以及任何修改或添加
到那个工作或其衍生作品，那是故意的
提交给许可方，由版权所有者将其纳入作品
或由个人或法定实体授权代表提交
版权所有者。为了这个定义的目的，“提交”
指发送的任何形式的电子，口头或书面通信
许可人或其代表，包括但不限于
电子邮件列表上的通信，源代码控制系统，
和问题跟踪系统，由管理或代表的管理
出于讨论和改进工作的目的许可人，但是
不包括显着标记的通信或其他通信
由版权所有者书面指定为“不贡献”。

“贡献者”是指许可人和任何个人或法人实体
代表出资方已经收到许可方和代表
随后纳入工作范围。

2.授予版权许可。服从条款和条件
本许可证，每个贡献者特此授予您一个永久的，
全球性，非排他性，免费，免版税，不可撤销
版权许可证复制，准备衍生作品，
公开展示，公开表演，再许可和分发
工作和源代码或对象形式的此类衍生作品。

3.授予专利许可。服从条款和条件
本许可证，每个贡献者特此授予您一个永久的，
全球性，非排他性，免费，免版税，不可撤销
（除本节规定外）专利许可制作，制作，
使用，出售，出售，进口和以其他方式转让作品，
该许可仅适用于可获许可的专利权利要求
由这样的贡献者，必然受到他们的侵犯
单独贡献或贡献（S）
与提交此类贡献的作品一起提交。如果你
针对任何实体（包括专利）提起专利诉讼
在诉讼中交叉索赔或反诉）指称该工作
或作品中包含的贡献构成直接
或共同的专利侵权，那么任何专利许可
根据本许可授予您的该作品应终止
截至提起诉讼的日期。

4.重新分配。您可以复制和分发
有或没有任何媒体的作品或衍生作品
修改，并以源或对象形式提供，只要您
符合以下条件：

（a）您必须提供任何其他收件人的作品或
衍生品制作本许可证的副本；和

（b）您必须使任何修改后的文件携带重要通知
指出你改变了文件；和

（c）您必须在任何衍生作品的来源表格中保留
您分发的所有版权，专利，商标和版权
来自作品来源形式的归属通知，
排除不属于任何部分的通知
衍生作品；和

（d）如果作品包含“通知”文本文件作为其部分内容
分发，那么您分发的任何派生工作必须
包含所含归属通知的可读副本
在此通知文件中，不包括那些不通知的通知
属于衍生作品的任何部分，至少有一部分属于衍生作品的任何部分
以下地方：分发通知文本文件
作为衍生作品的一部分；在源表单或
文件，如果与衍生作品一起提供；要么，
在由 Derivative Works 生成的显示中，如果和
无论此类第三方通知何时出现。内容
NOTICE 文件仅供参考，并仅供参考
不要修改许可证。您可以添加自己的归属地
在衍生作品中发布的通知，您一起分发
或作为工作通知文本的附录提供
这种额外的归属通知不能被解释
作为修改许可证。

您可以将您自己的版权声明添加到您的修改和
可能会提供额外的或不同的许可条款和条件
使用，复制或分发您的修改，或
对于任何此类衍生作品整体而言，只要您使用，
复制和分发工作是否符合
本许可证中规定的条件。

5.提交会费。除非您明确声明，否则，
任何有意提交列入作品的文稿
由您向许可方提交的条款和条件应符合

本许可证，没有任何附加条款或条件。
尽管如此，这里没有任何内容会被取代或修改
您可能执行的任何单独许可协议的条款
与许可方就此类贡献进行协商。

6.商标。本许可证不授予使用该交易的许可
许可人的名称，商标，服务标志或产品名称，
除非按照合理和习惯用于描述的要求
作品的来源和复制通知文件的内容。

7.免责声明。除非适用法律要求或
同意书面许可人提供的工作（和每个
贡献者在“按现状”的基础上提供其贡献）
没有任何形式的保证或条件，无论是明示还是暗示
暗示，包括但不限于任何担保或条件
不侵权，适销性或适用于 A 的适用性
特殊用途。您完全负责确定
使用或重新分配作品的适当性并承担任何责任
与您在本许可证下行使权限相关的风险。

8.责任限制。在任何情况下，也没有任何法律理论，
无论是在侵权行为（包括疏忽），合同还是其他方面，
除非适用法律要求（如故意和粗暴
疏忽行为）或书面同意，任何贡献者应为
向您承担损害赔偿，包括任何直接，间接，特殊，
偶然或因此而产生的任何性质的间接损害
本许可证的结果或不能使用或无法使用
工作（包括但不限于损失商誉，
工作中断，电脑故障或故障，或任何和所有
其他商业损失或损失），即使这样的贡献者
已被告知此类损害的可能性。

9.接受保修或附加责任。重新分配时
作品或其衍生作品，您可以选择提供，
并收取费用，接受支持，保修，赔偿，
或其他责任义务和/或与此一致的权利
许可证。但是，在接受这些义务时，您只能采取行动
代表您自己负责，而不是代表您
的任何其他贡献者，并且只有当您同意赔偿，
捍卫并保持每个贡献者免于任何责任
由于这种贡献者的理由所引起的，或由此产生的主张
您接受任何此类担保或额外责任。

条款和条件结束

附录：如何将 Apache 许可证应用于您的工作。

要将 Apache 许可证应用于您的工作，请附上以下内容
样板公告，附有括号“{}”的字段
换成你自己的识别信息。（不包括
括号！）文本应该包含在适当的
文件格式的注释语法。我们还建议 a
文件或类名称和目的说明包括在内
与版权声明相同的“打印页面”更容易
在第三方档案中识别。

贡献

贡献是非常鼓励！在做任何工作之前请先创建一个问题以避免失望。

我们创建了应该能让你迅速上手，如果你正在寻找一个标签 有趣的话题上手。

Python 约定

Python 代码应该遵循 pep-8 规范。

Python 2 和 3 交叉兼容性

为了确保 Python 2 和 3 之间的交叉兼容性，我们优先考虑 Python 3 惯例。请记住：

- 所有字符串文字都是 unicode 字符串
- 除法生成浮点数。使用`//`用于截断师
- 一些内置插件，例如 `map` 和 `filter` 在 Python 3，如果你想使用它们从导入的 Python 版本 3 人返回迭代器 `builtins`。否则，使用列表推导，这些列表推导在各个版本之间均匀工作
- 使用文件时 `io.open`，请使用内置代替内建函数 `open`
- 从下面的进口`__future__`是强制性的每个 Python 文件：`unicode_literals`，`print_function`，`division`，和 `absolute_import`

请参考此小抄了解如何编写与 Python 2 和 3 兼容的不同构造。

行为守则

Rasa NLU 遵守“ 贡献者公约行为准则”。通过参与，您需要维护这些代码。

文档

一切都应该妥善记录。要在本地测试您需要安装的文档

```
brew install sphinx
pip install sphinx_rtd_theme
```

之后，您可以使用以下方式编译和查看文档：

```
cd docs
```

```
make html
cd _build/html
python -m SimpleHTTPServer 8000 .
# python 3: python -m http.server
```

该文档将在 http://localhost:8000/上运行。

可以使用测试代码片段作为文档的一部分

```
make doctest
```

更改日志

这个项目的所有显着变化都将记录在这个文件中。该项目遵循从版本 0.7.0 开始的语义版本控制。

[Unreleased 0.12.0.aX] - master

注意

该版本尚未发布，正在积极开发中。

添加

变

固定

[0.11.4] - 2018-03-19

固定

- 谷歌分析文档调查代码

[0.11.3] - 2018-02-13

固定

- 在空间命名实体识别期间的资本化问题

[0.11.2] - 2018-02-06

固定

- 未在评估中分配实体的令牌格式

[0.11.1] - 2018-02-02

固定

- 更新日志文档格式
- 将新添加项目的项目加载到正在运行的服务器上

[0.11.0] - 2018-01-30

添加

- 非 ASCII 字符支持任何获取 json 转储的内容（例如通过 HTTP 端点接收的训练数据）
- 实体提取性能评估 `evaluation.py`
- 支持 spacy 2.0
- 意向分类与交叉效应评估 `evaluation.py`
- 支持将训练数据分成多个文件（仅限 Markdown 和 JSON）

变

- 从需求文件中删除- 如果你想安装应用程序使用`-e .pip install -e .`
- 修复非 `en` 语言的 httpduckling 解析
- 固定从降价训练数据文件解析实体

[0.10.6] - 2018-01-02

添加

- 支持 markdown 格式的示例样式注释

固定

- 防止资本化实体成为下层形式的同义词 - >大写

[0.10.5] - 2017-12-01

固定

- 从服务器读取令牌而不是数据路由器
- 固定读取服务器中没有日期名称前缀的模型

[0.10.4] - 2017-10-27

固定

- 码头图像构建

[0.10.3] - 2017-10-26

添加

- 支持新的对话流数据格式（以前的 api.ai）
- 改进了对自定义组件的支持（组件通过存储元数据中的类名存储，以允许在 Rasa NLU 注册表中未提及的组件）
- 语言选项来转换脚本

固定

- 修复 S3 中默认模型的加载。修复 # 633
- 训练失败时固定的永久训练状态 # 652
- 快速修复无“_formatter_parser”错误

[0.10.1] - 2017-10-06

固定

- 自述问题
- 改进的安装 py 欢迎信息

[0.10.0] - 2017-09-27

添加

- 以 Markdown 格式支持训练数据
- 科斯支持。您现在可以在配置文件中指定允许的 Cors 来源。
- HTTP 服务器现在由 Klein（Twisted）而不是 Flask 支持。服务器现在是异步的，但不再与 WSGI 兼容
- 改进的 Docker 自动构建

- Rasa NLU 现在可以与项目而不是模型一起使用。一个项目可以成为德国餐厅搜索机器人的基础或英语的客户服务机器人。模型可以被看作是项目的快照。

变

- 根项目目录已经稍微重新安排以清理新的 docker 支持
- 用于从 dict 创建解释器并使用文件中的元数据创建解释器 `Interpreter.create(metadata, ...)Interpreter.load(file_name, ...)`
- 将 `name` 参数重命名为 `project`
- 现在在 GitHub 页面上托管的文档： 文档
- 改编的远程云存储支持项目（向后不兼容！）

固定

- 修复了训练数据持久性。修复＃510
- 修正了通过 HTTP 接口进行训练时的 UTF-8 字符处理
- 在处理同义词时处理从 duckling 中提取的数字无效。修复＃517
- 只有在 mitie NER 期间，对未对齐的实体记录警告（而不是抛出异常）

[0.9.2] - 2017-08-16

固定

- 删除了不必要的 *ClassVar* 导入

[0.9.1] - 2017-07-11

固定

- 删除了过时的 `--output` 参数 `train.py`。 `--path` 改为使用。修复＃473

[0.9.0] - 2017-07-07

添加

- 增加测试覆盖率以避免回归（正在进行）
- 添加了正则表达式特征来支持意图分类和实体提取（`intent_entity_featurizer_regex`）

变

- 用 sklearn-crfsuite 替换现有的 CRF 库（python-crfsuite）（由于更好的 Windows 支持）
- 更新到 spacy 1.8.2
- 记录的请求的记录格式现在包括型号名称和时间戳记
- 使用模块特定的记录器而不是默认的 python 根记录器
- duckling 提取器的输出格式发生变化。该 `value` 字段现在包含来自 duckling 的完整值而不仅仅是文本（因此，这是一个属性，而不仅仅是文本）。现在包含粒度信息。
- 已弃用 `intent_examples` 且 `entity_examples` 部分训练数据。所有的例子都应该进入该 `common_examples` 部分
- 在 ner_crf 交叉验证和 sklearn 意图分类训练期间基于类别分布的权重训练样本
- 内部训练数据结构和流水线架构的大型重构
- numpy 现在是一个必需的依赖项

删除

- luis 数据标记器配置值（不再使用，luis 现在导出 char 偏移量）

固定

- 正确更新 travis 的工作服覆盖报告
- duckling 尺寸的持久性
- 未经训练改变默认响应 `intent_classifier_sklearn` 来自于 `"intent": None"intent": {"name": None, "confidence": 0.0}`
- `/status` 端点显示所有可用模型，而不仅仅是名称以*模型*开头的*模型*
- 正确地返回训练过程 ID # 391

[0.8.12] - 2017-06-29

固定

- 修复了缺少参数属性错误

[0.8.11] - 2017-06-07

固定

- 更新了 mitie 安装文档

[0.8.10] - 2017-05-31

固定

- 修复关于训练数据格式的文件

[0.8.9] - 2017-05-26

固定

- 正确处理被设置为的 response_log 配置变量 `null`

[0.8.8] - 2017-05-26

固定

- `/status` 端点显示所有可用模型，而不仅仅是名称以*模型*开头的*模型*

[0.8.7] - 2017-05-24

固定

- 固定范围计算 crf # 355

[0.8.6] - 2017-05-15

固定

- 修正 duckling 维度持久性。修复 # 358

[0.8.5] - 2017-05-10

固定

- 修复了 pypi 安装依赖关系（例如烧瓶）。修复 # 354

[0.8.4] - 2017-05-10

固定

- 修复了没有实体的 CRF 模型训练。修复 # 345

[0.8.3] - 2017-05-10

固定

- 修正了 Luis 的模拟并增加了测试来捕捉回归。修复 # 353

[0.8.2] - 2017-05-08

固定

- 上下文的深层复制 # 343

[0.8.1] - 2017-05-08

固定

- NER 训练重用了请求之间的上下文

[0.8.0] - 2017-05-08

添加

- ngram 字符特征（允许更好地处理词外词）
- 用更灵活的管道定义替换预先连线的后端
- 使用 sklearn 分类器 # 199 返回前 10 个意图
- 几乎所有公共函数的 python 类型注释
- 增加了定义实体同义词的替代方法
- 支持任意空间语言模型名称
- duckling 子成分为结构化实体提供标准化输出
- 有条件的随机场实体提取（用于实体标记的马尔可夫模型，具有低和中等数据的更好命名的实体识别，并且在大数据级别也类似）
- 允许命名经过训练的模型而不是生成的模型名称
- 动态检查不同组件的错误消息和缺失依赖关系的错误消息
- 支持使用多个实体提取器并将结果合并到下游

变

- 统一标记器，分类器和特征提取器来实现通用组件接口
- `src` 目录改名为 `rasa_nlu`
- 当以异形格式（api.ai，luis，wit）加载数据时，数据会被正确地拆分为意图和实体示例
- **配置：**
    - 添加 `max_number_of_ngrams`
    - 删除 `backend` 并添加 `pipeline` 为替换
    - 添加 `luis_data_tokenizer`
    - 添加 `duckling_dimensions`
- **解析器输出格式已更改**

  从 `{"intent": "greeting", "confidence": 0.9, "entities": []}`

  至 `{"intent": {"name": "greeting", "confidence": 0.9}, "entities": []}`

- **实体输出格式已更改**

  从 `{"start": 15, "end": 28, "value": "New York City", "entity": "GPE"}`

  至 `{"extractor": "ner_mitie", "processors": ["ner_synonyms"], "start": 15, "end": 28, "value": "New York City", "entity": "GPE"}`

  其中 `extractor` 表示最初发现实体的实体提取器，并且 `processor` 表示改变实体的组件，例如同义词组件。
- 骆驼式 MITIE 课程（例如 `MITIETokenizer` → `MitieTokenizer`）
- 模型元数据更改，请参阅迁移指南
- 更新到 spacy 1.7，并放弃了 spacy 组件的训练和加载功能（打破了现有的 spacy 模型！）
- 引入了与 Python 2 和 3 的兼容性

固定

- 正确地解析 `str` 附加到 `unicode` # 210
- 仅支持实体训练 # 181
- 解决了元数据和配置值 # 219 之间的冲突
- 在阅读 Luis.ai 数据（他们改变了他们的格式）时删除了标记。 # 241

[0.7.4] - 2017-03-27

固定

- 修改了重命名属性后的示例数据加载失败，即“KeyError：'实体'”

[0.7.3] - 2017-03-15

固定

- 在特殊字符上提取特定实体时的固定回归
- 对传递的语言实例进行固定的精细调整和实体识别

[0.7.2] - 2017-03-13

固定

- 有关从 python 调用 Rasa NLU 的 python 文档

[0.7.1] - 2017-03-10

固定

- mitie 标记化值生成 ＃207，谢谢@cristinacaputo
- 从更改的日志文件的扩展名`.json`来`.log`，因为包含的文本是不妥当的 JSON

[0.7.0] - 2017-03-10

这是一个主要的版本更新。请查看“ 迁移指南”。

添加

- 更新日志;）
- 在分类器训练期间使用多线程的选项
- 实体同义词支持
- 在测试期间正确创建临时文件
- mitie_sklearn 后端使用 mitie 标记和 sklearn 分类
- 可以选择微调 NER 模型
- 在 REST 服务器中构建多线程支持（例如使用 gunicorn）
- 多租户实现允许加载共享相同后端的多个模型

固定

- 错误传播失败的矢量模型加载（spacy）
- 在控制符号化期间转义特殊字符

[0.6-beta] - 2017-01-31