**使用Chatette为Bot构建数据集**

最近在研究使用开源框架RASA生成智能机器人的过程中，遇到一个无法回避的问题：在机器人进行自然语言处理时需要大量的预料库作为训练集，模型用训练集进行文本分类、命名实体识别、插槽填充或等任务，所以训练集越丰富，模型就会越准确，这些训练集是如何生成的呢？

# DSL

DSL的全称是domain-specific language,中文意思是特定于域的语言，是一种针对特定应用领域而设计使用的计算机语言。

常见的DSL有:Shell,make语言,ant语言,maven语言，rpm语言，dpkg语言，awk语言等。

DSL是专门为解决特定域中的问题而创建的，并不旨在解决其外部的问题。

本文简要介绍一个为智能机器人创建复杂预料库的DSL工具-Chatette。

## **Chatette**

Chatette实现了一种特定于域的语言，允许你定义模板来生成大量句子，然后将这些句子以Rasa NLU的输入格式保存。

## **定义模板文件**

在chatette中你可以创建一个或多个模板文件供Chatette解析，模板文件名后缀是.chatette。

示例：Dsmart\_bot的训练集chatette文件部分内容

|  |
| --- |
| // 根据uid查询健康分数  %[query\_score\_with\_info]('context': '根据uid或实例名查询健康分')  {我~[需要]/~[请求?]}~[查询]{~[uid]~[是]@[uid]/~[ins\_name]~[是]@[ins\_name]}的~[健康分]  %[健康分数]('context': '请求查询健康分(默认不提供任何信息)')  ~[健康分]  {我~[需要]/~[请求?]}~[查询]~[健康分]  {我~[可以]/~[可以]~[帮助]}~[查询]~[健康分]吗？  %[日检报告]('context': '请求查询日检报告(默认不提供任何信息)')  ~[日检报告]  {我~[需要]/~[请求?]}~[查询]~[日检报告]  {我~[可以]/~[可以]~[帮助]}~[查询]~[日检报告]吗？  ~[uid]  uid  数据库uid  数据库id  oracle uid  ~[ins\_name]  实例名  oracle 实例名  实例  instance name  ~[健康分]  健康分  健康评分  健康得分  健康分数  健康度得分  健康模型得分  ~[日检报告]  日检  日检报告  |data.chatette |

这段代码输出格式独立于DSL语言,推荐使用换行符分隔的格式流,生成json格式文件。Json训练集样本的规模可以达到成千上万行。

|  |
| --- |
| {  "rasa\_nlu\_data": {  "common\_examples": [  {  "entities": [  {  "end": 20,  "entity": "ins\_name",  "start": 9,  "value": "instance\_01"  }  ],  "intent": "query\_score\_with\_info",  "text": "请你帮我查询实例为instance\_01的健康得分"  },  {  "entities": [  {  "end": 18,  "entity": "ins\_name",  "start": 7,  "value": "instance\_01"  }  ],  "intent": "query\_score\_with\_info",  "text": "我想查查实例为instance\_01的健康得分"  },  {  "entities": [  {  "end": 17,  "entity": "uid",  "start": 8,  "value": "210100002"  }  ],  "intent": "query\_score\_with\_info",  "text": "查数据库uid为210100002的健康分"  },  {  "entities": [],  "intent": "健康分数",  "text": "看一看健康模型得分"  },  {  "entities": [],  "intent": "健康分数",  "text": "可以替我看健康度得分吗？"  },  {  "entities": [],  "intent": "健康分数",  "text": "我能查健康评分吗？"  },  ... ...  ... ...  ... ...  ],  "entity\_synonyms": [],  "regex\_features": []  }  } |

## **语法**

模板文件的每一行可以是下面4种格式的一种：

1. 空行和注释

注释行以//开头，不会被chatette解析。

1. 单元声明开头

以（%,@或~）特殊字符开始的无缩进行。

1. 单元声明内容

声明内容必须以同样的方式缩进，每一行称为声明单元的规则。

1. 包含其他文件

告诉编辑器包含其他chatette文件。

**总结**

通过chatette我们可以为机器人迅速构建大规模的复杂预料库，可以为我们节省大量的时间用于构建业务场景，在减少工作量的同时也提高了效率。

# chatito

# 语法说明

Entities(实体):具有特殊行为的特殊关键字，用于声明句子组合。

实体参数:可选的键值，可以在intent和slot定义中声明。

概率运算符:在句首声明的可选关键字，用于控制概率。

句子:实体定义后的4行空格缩进文本行。

实体是定义关键字的一种方法，这些关键字封装句子变体，并为它们附加一些属性。有三种类型的实体:intent、slot和alias。

Intent(意图)

以%[]开头，无缩进，[]中为Intent名称。

不允许不同的intent名称重复。

每个意图都是生成的入口点，意图是添加到其中定义的句子中的分类标记。

|  |
| --- |
| import import.chatito  %[greet]('training': '2', 'testing': '1')  ~[hi] ~[how are you?]  ~[hi] @[name?]  #name为插槽  @[name]  Janis  Bob  #import.chatito文件如下  import nestedimport.chatito  ~[hi]  ~[saludos1] ~[saludos2?]  ~[saludos2]  #nestedimport.chatito文件内容如下  ~[saludos1]  hola1  hi1  hihi1  ~[saludos2]  hola2  hhi2  hello2 |

前面的例子将生成所有可能的问候独特的例子。

但是有些情况下不需要生成所有话语，或者当我们想要为生成的话语附加一些额外的属性时，这就是实体参数可以提供帮助的地方。

实体参数是用括号内的实体定义声明的逗号分隔键值。每个实体参数由一个键组成，后跟：符号和值。参数键或值只是用单引号或双引号括起来的字符串，括号和逗号之间的可选空格是允许的，格式类似于ndjson但仅适用于字符串值。

默认情况下，意图定义可以期望训练和测试参数键在定义时用于声明为给定意图生成的唯一示例的最大数量，并将它们分成两个数据集。训练数据集将用于训练NLU模型，用测试数据集来评估模型的准确性，并使用从未训练过的示例。

创建测试数据集不是必须的，但重要的是要了解模型的准确性，以检测过度拟合以及和先前的精度进行比较。生成器将首先填充训练数据集，然后测试数据集直到达到两个值的总和，每个值必须> = 1。

greet intent可以生成很多样本，但声明仅请求3个训练样本，其将含2个greet intent训练集和测试数据集1的话语。

默认情况下，其他实体参数将被忽略，其功能取决于数据集生成器/适配器，这意味着每个适配器可以在其自己的上下文中以不同方式使用其他实体参数（例如：Rasa / Snips适配器可能需要不同的实体参数）。

嵌套实体：在intent中定义的句子可以引用槽和别名实体。

**Slot（插槽）**

插槽实体形式：@ [插槽名称]开头定义。

插槽名称长度应至少为1个字符。

不应允许重复插槽名称定义。

从输出角度来看，slot是在生成的句子中添加相关单词的标记。

句子中引用的插槽实体可以有？引用名称末尾的符号。

例如：@ [name？]，此种情况下的？表示插槽组合是可选的，可以在生成时省略。

插槽在其定义中提供称为变体的特定属性。

变化：有些情况下，插槽组合只在给定的上下文中有意义，变体允许在不同的上下文中将一个插槽映射到不同的句子。

|  |
| --- |
| %[ask\_for\_delivery]  my parcel should be delivered in @[delivery\_time#time\_in\_hours]  my parcel should be delivered @[delivery\_time#relative\_time]  @[delivery\_time#time\_in\_hours]  3 days  5 hours  @[delivery\_time#relative\_time]  as fast as possible  quickly |

在此示例中，两个组合都映射到delivery\_time词槽，但生成的句子仅生成它们有意义的变体上下文。

插槽定义也可以包含实体参数，但没有默认参数键。默认情况下会忽略实体参数，它们的功能取决于数据集适配器，这意味着每个适配器可以在其自己的上下文中以不同方式使用实体参数。

例如：Rasa / Snips适配器可能需要不同的实体参数，例如预构建日期解析，或文本值别名映射。

嵌套实体：插槽内定义的句子只能引用别名实体。

**别名**

别名实体由〜[别名名称]定义开始，无缩进。

别名只是单词的变体，不会生成任何标记。

默认情况下，如果别名被引用但未定义别名，就像示例中的how are you，它只使用别名键名称，这使单词可选但没有添加定义新的额外代码行。

|  |
| --- |
| %[greet]  ~[hi] ~[how are you?]  ~[hi]  hi  hey |

当在槽定义中引用别名，并且它是槽句的唯一标记时，默认情况下，生成器将生成的别名value标记为别名key的同义词。

别名定义不允许声明实体参数。

嵌套实体：别名内定义的句子可以引用插槽和其他别名，但要防止递归循环。

句子生成概率

Chatito的工作方式就像从可能组合的云中提取样本，但一旦句子定义开始变得越来越复杂，最大可能的组合可能性会呈指数级增加，从而导致生成器最有可能选择具有更多可能组合的句子的问题，并省略一些在数据集中可能更重要的句子。

要对生成器原理进行一些控制，可以使用概率运算符。

句子概率算子由\* [产生句子的概率]来定义。概率运算符内的值必须是1到100之间的整数，实体定义中所有概率运算符的总和不应超过100。

|  |
| --- |
| %[greet]('training': '100', 'testing': '100')  \*[50] ~[phrase1]  \*[30] ~[phrase2] ~[phrase3?]  ~[another phrase] ~[something] ~[something else] |

第一句有50％的几率。第二句将产生30％的话语，剩余的20％将来自所有句子的剩余可能性。

注意：使用概率运算符时要小心，如果句子达到其唯一生成值的最大数量，它将开始生成重复项并可能减慢可能过滤重复项的生成器。

数据生成

数据生成的入口点是每个intent定义的intent定义： 如果意图未指定“training”或“testing”参数，则生成所有可能的唯一组合并将其添加到训练数据集。

遵守opreator声明的概率 。

为训练数据集生成唯一组合，直到达到“训练”参数编号，然后直到达到测试数据集的“测试”参数编号。

应该防止递归循环引用。

Chatette

简单intent定义

|  |
| --- |
| %[greeting](2)  hello  hi  Hi! |

生成的输出文件中将有3个字符串的其中2个。

Slot

生成规则中的槽在Rasa NLU中称为实体，即特定变量的有限值集，其中不同值表示不同的事物。

|  |
| --- |
| @[operating system]  Linux  Windows  macOS  FreeBSD/OpenBSD  I use @[operating system]. |

生成的话语中， I use Linux占25%  I use Windows 25% , I use macOS占25% ， I use FreeBSD/OpenBSD 也25%。

如果需要多个不同的文本来表示相同的值，需要使用=语法，即在规则的末尾添加一个相等的实体值（等号周围的任何空格都将被忽略）。这样，您可以将不同的字符串映射到同一实体值。

例如，希望Firefox，FireFox和ff都映射到值Firefox（从而获得为Rasa NLU生成的JSON数组），无论生成哪个字符串都可以按如下方式定义插槽

|  |
| --- |
| @[browser]  Firefox  FireFox = Firefox  ff=Firefox  Chrome/Chromium  //... |

同样可以使用选择修饰符/

|  |
| --- |
| @[browser]  {Firefox/FireFox/ff} = Firefox  Chrome/Chromium  //... |

如果对规则使用特殊符号/（例如Firefox = /），则在选择此规则时，插槽的值将是规则中第一个子规则的“名称”（即使它不生成任何东西）。这个“名称”是单词的单词本身，单词组中单词的字符串以及单位引用的单位名称。作为选择，这个“名称”是没有修饰符的花括号内的字符串。因此，规则~[this] is a rule = /在槽定义内部会生成文本this is a rule

（例如）并且具有值this。

如果生成规则内需要使用=，可以用转义符号\=。

修饰符

对于单元声明和子规则，可以添加修饰符来在遇到生成器时更改生成器的行为。仅针对当前子规则/单元声明更改生成行为。

另请注意，简单的单词不能使用任何修饰符。

大小写生成：生成器在生成单元的引用中，在第一个字母大写或小写之间随机选择，修饰符用＆符表示，位于单元声明启动器或子规则的开括号[（或{for choices））之后。

例如，[＆hello]将在50％的时间内生成hello，在其余时间生成Hello。

|  |
| --- |
| @[&doctor]  doctor  Dr. |

这个规则将生成doctor,Doctor,Dr. ,dr.四个值各占25%。

随机生成：添加问号？在关闭括号之前（或者用于选择）的子规则告诉生成器随机决定它是否应该忽略该子规则，因此它应该生成一个字符串，或者更确切地说是什么。

除了选择之外，可以在问号后面为此随机生成提供标识符;具有该随机生成标识符的每个子规则将一起生成或根本不生成（而不是生成其中一些而不生成其他子规则）。

例如，规则hey [you?] 将生成 hey 占50% 和 hey you 占另外50%;

规则 Hi [I'm a?rand name] pretty [test?rand name] 将生成 Hi I'm a pretty test 占50% ， Hi pretty 占另50%, 但一定不会生成Hi I'm a pretty 或Hi pretty test。

sentence

变体命名：添加一个标签后跟一个特定的名称是告诉解析器正在制作单元的变体。此单元变体可以在规则中作为正常单元引用，但也可以引用没有变化的单元，这将引用该单位的所有变体。通常用于制作相同别名的单数和复数味道，并且在单数或复数无关紧要的地方使用没有变化的单元。

例如，如果定义了以下别名：

|  |
| --- |
| ~[alias#singular]  alias  ~[alias#plural]  aliases |

子规则 ~[alias#singular] 将生成alias, ~[alias#plural] 将生成aliases ，~[alias] 将生成50%的 alias  和50%的 aliases。

有时需要让一个单元的某些部分可以在以后填充。

有时需要能够引用某个单元，但是在此引用生成的字符串中包含某个字符串。 此修饰符使用$表示，该符号放在单位声明启动器的末尾（在结束方括号之前），后跟参数的名称。然后，可以在本单元声明的部分或全部规则中放置相同的字符串（即$后跟参数的名称）。 引用此单元时，使用$后跟此参数应采用的值将使用此值代替规则中的参数。

|  |
| --- |
| ~[greetings$NAME]  Hi $NAME  Hello $NAME! |

using the reference ~[greetings$John] in another rule will generate Hi John or Hello John!, while using the reference ~[greetings$Elvis] will generate Hi Elvis or Hello Elvis!. Note that using simply the reference ~[greetings] will generate Hi $NAME or Hello $NAME! without warnings

参考：

<https://github.com/rodrigopivi/Chatito/blob/master/spec.md>

<https://github.com/SimGus/Chatette/wiki/Quickstart>

<https://github.com/SimGus/Chatette/wiki/Generation-rules>