[用户指南](http://www.opendial-toolkit.net/user-manual) >

循序渐进的例子

|  |  |
| --- | --- |
| 假设你想为一个机器人创建一个简单的对话系统，可以指示机器人在四个方向上移动：左，右，前，后。当不确定时，机器人应该要求用户重复说明。概率规则允许我们直接设计这样一个对话系统。  我们在这个例子中关注对话域的规范，以处理用户输入并根据对话情况选择最相关的动作。当然，真正的机器人还需要包含用于实际语音识别，语音合成和电机控制的不同模块。[语音识别和综合](http://www.opendial-toolkit.net/user-manual/speech-recognition-and-synthesis)以及[外部模块](http://www.opendial-toolkit.net/user-manual/external-modules)部分介绍了这些模块的集成。   |  | | --- | | ***符号约定***：  在这个例子中（和OpenDial一般），我们将标签与后缀用户的所有变量\_u（带u代表“用户”）和相关系统的所有变量\_m  （用m代表“机器”）。因此变量 u\_u是用户话语u\_m的默认标签，而系统话语的默认标签。这些标签可以在系统设置中更改。  类似地（并且如在对话管理文献中常规使用的），用户对话行为被表示a\_u，而系统行为被表示a\_m。 |   **一般骨架**  我们首先创建一个新的对话域（Domain > New在菜单栏中）。如果我们转到域编辑器选项卡，我们会看到一个空的域规范：  <domain>  <!-- the domain specification will go here -->  </domain>  每个对话域由一组（规则结构）*模型组成*。模型本质上是一个概率规则的集合，同时还有一个触发器变量，用于指示何时应用规则。  在我们的例子中，我们想要定义一个当观察到新用户话语时触发的模型。因此，我们将构造一个变量u\_u作为触发器的模型。域规范变为：  <domain>   <model trigger="u\_u">  <!-- the rule(s) for this model will go here -->  </model>  </domain>  每个模型内部都是一系列概率规则。[[1]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B1%5D)可以编码两种不同类型的规则：   * *概率规则* 表示一些状态变量（*规则*的“输入变量”）如何影响其他一些状态变量（“输出变量”）的值。换句话说，他们编码形式为P（**O | I**）的条件概率分布，其中**I**代表输入变量，**O**代表输出变量。 * *效用规则*根据特定的输入变量来表达特定动作（从系统角度）的效用。换句话说，它们编码U（**A** | **I**）形式的效用函数，其中**I**表示输入变量和**A**动作变量。   **第一条规则**  由于我们想将用户话语映射到特定的系统操作，我们的第一条规则将是一条实用规则。我们可以按照以下方式对用户话语u\_u和系统话语u\_m之间的映射进行编码：    <rule>       <case>         <condition>           <if var="u\_u" value="turn left" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, turning left!" />         </effect>       </case>       <case>         <condition>           <if var="u\_u" value="turn right" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, turning right!" />         </effect>       </case>       <case>         <condition>           <if var="u\_u" value="move forward" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, moving forward!" />         </effect>       </case>       <case>         <condition>           <if var="u\_u" value="move backward" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, moving backward!" />         </effect>       </case>   </rule>  正如我们所看到的，每个规则都由一个有序的案例 元素列表组成。每个案例都与一个特定的条件和一组效果相关联（尽管在这个特定的规则中，每种情况下只有一个效果）。该规则可以这样理解：  **如果**用户输入u\_u等于“ 左转 ”，**那么**  u\_m =“ OK，左转 ” 的效用设置为1。  **否则，如果**用户输入u\_u等于“ 右转 ”，**那么**  u\_m =“ OK，右转 ” 的效用设置为1。  **否则，如果**用户输入u\_u等于“ 向前移动 ”，**那么**  u\_m =“ OK ”的效用，向前移动设置为1。  **否则，如果**用户输入u\_u等于“ 向后移动 ”，**那么**  u\_m =“ OK，后退 ” 的效用设置为1。  **其他**  没有设置实用程序。  我们现在可以运行OpenDial，打开我们设计的域，然后在聊天窗口中键入例如“ 左转 ”。系统响应应该是“ OK，左转！ ”，因为系统会自动选择具有最高效用的动作。  我们还可以单击状态查看器并检查当前对话状态（以贝叶斯网络的形式）以及状态更新期间的中间状态。  **语言理解模型**  当前对话领域的一个缺点是其可能的用户输入的严格范围。用户话语，如“ 向左转 ”或“ 现在请向前移动 ”等，都被系统忽略。尽管我们原则上可以直接枚举效用规则中的所有可能输入，但更原则的方法是编写将用户话语转换为用户对话行为的逻辑表示（表示为a\_u）的概率规则，然后让该实用新型在这个逻辑表示上运行。  因此，我们将用一个单一概率规则和触发变量u\_u编写一个新模型：   <model trigger="u\_u">   <rule>     <case>      <condition operator="or">       <if var="u\_u" value="turn \* left" relation="contains"/>       <if var="u\_u" value="move \* left" relation="contains"/>       <if var="u\_u" value="go \* left" relation="contains"/>     </condition>     <effect prob="1">       <set var="a\_u" value="Request(Left)" />     </effect>     </case>     <case>     <condition operator="or">       <if var="u\_u" value="turn \* right" relation="contains"/>       <if var="u\_u" value="move \* right" relation="contains"/>      <if var="u\_u" value="go \* right" relation="contains"/>     </condition>     <effect prob="1">       <set var="a\_u" value="Request(Right)" />     </effect>    </case>    <case>     <condition operator="or">       <if var="u\_u" value="move \* forward" relation="contains"/>       <if var="u\_u" value="go \* forward" relation="contains"/>       <if var="u\_u" value="go \* straight" relation="contains"/>     </condition>     <effect prob="1">      <set var="a\_u" value="Request(Forward)" />     </effect>    </case>    <case>     <condition operator="or">       <if var="u\_u" value="move \* backward" relation="contains"/>       <if var="u\_u" value="go \* backward" relation="contains"/>     </condition>     <effect prob="1">       <set var="a\_u" value="Request(Backward)" />     </effect>    </case>    <case>     <effect prob="1">       <set var="a\_u" value="None" />     </effect>   </case>   </rule>  </model>  在这个模型中有一些值得注意的事情。首先，规则条件有点复杂。每个条件都被编码为基本条件的分离，如属性operator = or所示。  基本条件的一个例子是      <if var="u\_u" value="turn \* left" relation="contains" />  只要在用户话语u\_u的字符串内发现模式转向\*左边，就满足该条件。属性relation =“contains” 指示条件检查该模式是否被包括为完整话语的子字符串（换句话说，它执行部分匹配）。该\*号表示通配符，可以捕获任何序列。[[2]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B2%5D)  效果本身指定必须更新输出变量（在这种情况下，用户对话操作a\_u）的方式。在这种情况下，所有效果都是确定性的。然而，我们稍后会遇到非确定性的规则（即它们包括几种具有不同发生概率的替代效应）。  **动作选择和生成模型**  不是直接将系统话语硬编码为系统动作，而是通过两个步骤对系统决策进行分解通常更合适：   1. 选择下一个动作的高级逻辑表示（表示为a\_m） 2. 为这个合乎逻辑的行为选择最好的语言实现。   行动选择模型与前面介绍的实用新型非常相似，除了它在用户对话操作a\_u上操作并选择高级别操作a\_m：  <model trigger="a\_u">     <rule>       <case>         <condition>           <if var="a\_u" value="Request(Left)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Left)" />         </effect>       </case>       <case>         <condition>           <if var="a\_u" value="Request(Right)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Right)" />         </effect>       </case>       <case>         <condition>           <if var="a\_u" value="Request(Forward)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Forward)" />         </effect>       </case>       <case>         <condition>           <if var="a\_u" value="Request(Backward)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Backward)" />         </effect>       </case>     </rule> </model>  生成模型可以通过一个单一的实用规则轻松构建：  <model trigger="a\_m">   <rule>       <case>       <condition>         <if var="a\_m" value="Move(Left)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, turning left!"/>       </effect>     </case>     <case>       <condition>         <if var="a\_m" value="Move(Right)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, turning right!"/>       </effect>     </case>     <case>       <condition>         <if var="a\_m" value="Move(Forward)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, moving forward!"/>       </effect>     </case>     <case>       <condition>         <if var="a\_m" value="Move(Backward)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, moving backward!"/>       </effect>     </case>   </rule> </model>  **澄清策略**  面对噪音和不确定性，目前的领域缺乏稳健性。例如，如果系统以0.1的低概率观察诸如“ u\_u =向左移动 ” 的用户话语（这可以在聊天窗口中通过在话语的末尾添加括号中的概率来测试），它将忽略该指令非常不确定，并选择Move（Left）作为下一个操作。  更好的方法是只在达到某个概率阈值时执行操作。可以将以下规则添加到动作选择模型中：    <rule>       <case>         <effect util="-0.5">           <set var="a\_m" value="Move(\*)"/>         </effect>       </case>   </rule>  该规则将使所有Move（\*）动作的效用减少0.5。换句话说，这意味着这些行为只有在概率高于0.5时才会执行。  我们还可以添加另一个系统动作AskRepeat来请求用户在面对不确定性时重复说话：    <rule>       <case>         <effect util="0.2">           <set var="a\_m" value="AskRepeat"/>         </effect>       </case>   </rule>  生成规则还应该扩展为另一种情况：  <!-- ...-->   <case>       <condition>         <if var="a\_m" value="AskRepeat"/>       </condition>       <effect util="1">         <set var="u\_m" value="Sorry, could you repeat?"/>       </effect>     </case>  </rule>  我们可以在OpenDial用户界面中测试生成的对话域，并验证生成的系统行为：  **先前的分配**  包含在当前对话域中的AskRepeat操作并不十分复杂。它只是要求用户重复，但不会在轮到时“积累”证据。假设例如对话摘录：  用户：    前进（0.65 ） 系统：对不起，你能重复吗？用户：    前进（0.65 ）    在当前的对话域中，系统将再次要求用户重复。理想情况下，这两个话语中的最高假设是“ 向前移动” 的事实应该为该系统提供对“ 向前移动 ” 的假设增加的信心。  我们可以编写一个规则来编码常识性假设，即在用户需要时可能会重复他/她的话语：  <!-- Prediction on the next user action --> <model trigger="a\_m">     <rule>       <case>         <condition>           <if var="a\_m" value="AskRepeat" />         </condition>         <effect prob="0.95">           <set var="a\_u^p" value="{a\_u}" />         </effect>       </case>     </rule> </model>  上述规则规定，当系统请求用户重复该指令时，下一个用户对话行为被预测为与当前的对等行为相同，概率为0.95（剩余的0.05包括用户决定说另外一些事情的情况）。  为了区分未来事件（在这种情况下，下一个对话行为）与实际观测值的这种预测，OpenDial依赖于预测变量用上标^ p表示的约定。变量X ^ p因此表示对将来要观察的变量X的预测。[[3]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B3%5D)  读者还应该注意，效果值写为{a\_u}。此值是一个*参考*的变量的当前值a\_u。大括号{}很重要：没有它们，效果会简单地声明a\_u ^ p必须设置为字符串“a\_u”，而不是引用变量所表示的值。  由于这一规则提供了对下一次对话行为的事先分配，对话系统能够积累证据并选择正确的行动来执行：  **使用通用量词**  当前对话域中的一些规则呈现出循环模式：例如，系统动作的效用规则指出，如果用户对话行为具有Request（*某个动作*）的形式，则系统可以执行相应的动作Move（*某些行动*）与效用1。  通过使用逻辑量词可以大大丰富概率规则的表达能力。规则条件和效果的确可以部分低估并包含自由变量。换句话说，规则指定的条件和效果之间的映射对于这些自由变量的每个可能的值赋值都是重复的。[[4]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B4%5D)  因此，选择下一个系统操作的实用规则可以简化为以下内容：    <rule>       <case>         <condition>           <if var="a\_u" value="Request({X})" />         </condition>         <effect util="1">           <set var="a\_m" value="Move({X})" />         </effect>       </case>   </rule>  大括号被用来表示自由变量X。在运行时，OpenDial将为自由变量确定可能的赋值集合（称为*基础*），并为这些基础中的每一个复制规则。请注意，自由变量标签（如X）不得与现有状态变量的标签冲突。  **参数**  当前域的概率和效用都是手工制作的。尽管这种手工方法在特定情况下可能运作良好，但它仍然容易受到人为错误和不准确的影响。例如，AskRepeat动作的效用（当前设置为0.2）或者在这样的请求之后用户重复的概率（设置为0.95）仅仅是被告知的猜测，并且实际的交互可能很好地偏离这些期望值。  更原则的是，数据驱动的方法是将这些规则与必须根据数据估计其值的参数相关联。OpenDial中的参数都可以替代概率和实用程序。  由于OpenDial采用贝叶斯方法进行参数估计，因此每个参数必须与其在通常连续的值范围内的先验分布关联。为此，可以使用几种类型的参数分布，如均匀分布，高斯分布和狄利克雷分布。[[5]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B5%5D)  我们可以为我们的域创建两个参数：   * 一个用于AskRepeat 操作实用程序的参数。此参数的合理先验分布是以0为中心的高斯分布。就本例而言，我们应将此分布设置为〜N（0,5）。 * 用户重复概率的一个参数。由于我们可能期望用户在大多数情况下遵守系统请求，因此我们可以将此概率分布编码为Dirichlet分布〜Dir（3,1）。   这些先前的参数分布在域规范的顶部指定：   <parameters>      <variable id="theta\_repeat">       <distrib type="gaussian">         <mean>0</mean>         <variance>5</variance>       </distrib>     </variable>      <variable id="theta\_repeatpredict">       <distrib type="dirichlet">         <alpha>3</alpha>         <alpha>1</alpha>       </distrib>     </variable>   </parameters>  最后的修改是用它们的参数替换概率规则中的固定值：     <rule>       <case>         <effect util="theta\_repeat">           <set var="a\_m" value="AskRepeat" />         </effect>       </case>     </rule>  ...        <rule>       <case>         <condition>           <if var="a\_m" value="AskRepeat" />         </condition>         <effect prob="theta\_repeatpredict[0]">           <set var="a\_u^p" value="{a\_u}" />         </effect>       </case>     </rule>  请注意，由于Dirichlets是多变量分布，所以第二个规则的参数必须对分布的维数（在这种情况下为第一维：[0]）进行索引。  可以通过状态监视器检查两个参数的分布情况：  必须根据对话数据优化参数。这将在[参数估计](http://www.opendial-toolkit.net/user-manual/parameter-estimation)部分中介绍。  **最终域**  对话域的完整XML规范可以在[domains / examples / example-step-by-step\_fixed.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-step-by-step_fixed.xml)（没有未知参数）和[domains / examples / example-step-by-step\_params.xml中找到](https://github.com/plison/opendial/blob/master/domains/examples/example-step-by-step_params.xml)（带有未知参数）。  [1]概率规则被用作涵盖概率和效用规则的总称。  [2]参见部分[字符串匹配](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#4._String_matching)用于在OpenDial实现的字符串匹配的功能的更多细节。  [3]见[Lison（2014年）](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf)，第9页。详情请参阅78-79。  [4]见[Lison（2014年）](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf)，第5页。67-68和74-76的细节。  [5]见[Lison（2014年）](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf)，第9页。91-96了解更多详情。 |