拼音输入法更新文档

Z输入法, V 2.0

周昊一,张绍文,郑健,张昊,周佳雨

注:本文档是多人合作完成的,因此语言风格上会有一些差异,可能还有一些重复,其中一至五是一部分,六是一部分,七是一部分.组员学号,具体分工等在八中有说明.

# 更新内容

此次更新主要是涉及到语言模型方面.基于HMM模型训练并建立了一个语言模型,这个语言模型主要的用途是更好地实现简拼功能.同时,在语言模型的训练过程中,也获取了新的带有权重数据的字典与词典,进一步提升了输入过程中的准确率.

此外,还对原有的一些方面进行了一些优化.例如在音节切分方面,再加入了反向最大匹配,用于处理类似于”xian->xi’an”这样的情况.另外还对运行速度进行了一些优化,由于HMM模型的加入,原本性能会有显著下降,通过这些优化尽可能的将性能维持在原有水平.

# 基于语料库的新词库生成

## 语料库

本次所使用的语料库是2007~2011年5年的人民日报的数据.总计196051篇文章,892M的文本.

语料库是本次整个工作的基础,新的字库,词库,以及语言模型都是在它的基础上产生的.因此语料库的语言偏向性即代表了最后整个输入法的偏向性.

由于人民日报中的文章用词都比较正式,鲜有口语并且多涉及人名地名,因此最后可以看到生成的整个语言模型对于口语化的语句输入不能够很好的处理,但是对于比较正式的语句能够较为顺畅的输入.

## 新的字库与词库的挖掘与生成

在上一个版本中,所使用的字库与词库都是人工搜集或生成的.具体的讲,字库是我们在半手工地生成的完整字库,并对其中一些字的权重作了一定的人工标注.而词库则是从网上收集的高频词词库,由于词数太多,做人工标注不现实,并且也无法保证标注结果的有效性,因此没有做标注.

因此,在这一版本中,我们希望能通过语料库来得到一个更为完善,有数据支撑的词库.

我们的做法是,先对文本进行分词,再对分好词的文本进行统计,记录每个出现了的词的词频,同时也记录每一个词的词频.

接下来进行一次筛选,将词频过低的词去除,因为这些词频过低的词显然不是常见的词语.根据观察,通常低词频的词都是人名与地名,这都是我们所不希望出现在我们的核心词库中的.

随后我们根据词(单字也算作词)的词频f算出它的权重w.虽然可以直接令w=f,但是实际上高词频的词与低词频的词之间词频差距太大,在计算概率时会因为浮点数精度问题导致得到为0的概率,因此需要某种手段来缓和高频词和低频词之间的差距.我们所选用的方法是取对数,w=ln(f+1),之所以+1是为了保证w不为0..

最后,将得到的词库输出为我们所需要的格式.

# 基于HMM的简拼查找

# HMM模型简介

# HMM模型的训练

首先我们描述一下输入法中所使用的HMM模型:

用户的输入表示一系列用户输入状态,例如(xin),(xing)就是两个不同的状态.每个用户输入状态(即拼音)对应一个中文字串.如假设在某次输入时(xin)对应的中文字串是(新),但在其他时候(xin)还可能对于(心),(信)等.

在这里表示的是可观测的外部状态,而表示的是不可观测的内部状态.不同的内部状态有可能对应相同的外部状态,不同的外部状态也可能对应相同的内部状态(比如简拼的情况).而我们的目标就是对一个给定的,要能得到一个内部状态的序列,其中内部状态是按照可能性的大小降序排列的.

对于全拼

# HMM模型的使用

# 其他改进与优化

## 简拼的完整功能描述

## 目前的实现

第一点实现了部分.方法是通过音节切分,找到不完整的音节,并进行猜测(当前是将所有情况全部试一遍).但是用这种方法基本只能做到一个音节的还原,因为平均每个简拼对应的全拼大概有10个,如果要还原两个,搜索空间就是100,而平均每个全拼对应的字词至少有20个,这样的话候选词就有20\*100=2000个之多,从效率上来说无法接受(当前再只还原一个的情况下在有些时候都已经能明显感觉到卡顿,可能是因为JNI在C++和Java交互时的效率问题,毕竟必须要进行一次数据的转换,从C++的字符串变成Java的对象,以后会考虑将这部分也放到C++中,尽量减少C++和Java的交互).

当然即使是在PC机上,这种方法也是不太可取的,因为每多一个还原的音节,搜索空间就上升大约10倍.所以以后的版本的改进肯定是要基于一个新的预测词库进行预测,这个词库的生成还需要对现有词库以及大量的语料库进行分析,找到一些高频的组合.

第二点则是已经实现了,这一点在目前的数据结构下很好实现,只要把每个词的首字母简拼也作为key再存一遍就可以了.

第三点实在第一点的基础上实现的,就是简单的把所有的组合都搜索一遍,存在的问题也一样,就是搜索空间增长太快,需要和第一点做同一个方向的改进.

第四点目前没有实现,目前估计可以通过对字的code进行反查的方式进行实现.即先通过简拼查出所有的可能,如上面的例子中就是先通过”zhrm”查出所有可能的词,再对”zhrmo”进行切分(有两种切法,分别对应前面查出来的不同字数的候选词),再反查到所有候选词中对应”mo”这一位的字的code(可以通过一个额外的字典表来实现),将所有能匹配的候选词选出.

# 关于Android平台

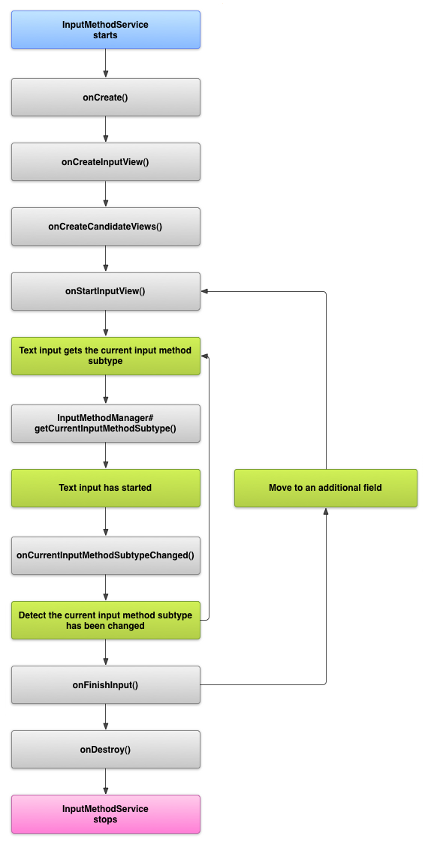


图2. 输入法的生存周期

在这个生存周期中,我们的输入法内核位于onStartInputView()到onFinishInput()这个循环中.前面的三个步骤都是和UI初始化有关的.

在初始化时,我们建立好用于拼音切分的树以及词典.当用户每次输入一个字母时,需要对当前的整个输入串进行一次处理.处理流程为

1.切音,将用户的输入切成音节序列.

2.填充(预测),如果切音时发现了不完整的音节

3.查词,将所有的输入情况转换为候选词.这里的所有输入情况是指,对于每一个长度为n的音节切分,需要进行n次查找,每次的key是这个音节切分的一个长度为k(k=1~n)的子序列.比如对于”di’qiu’ren”这个切分,需要查找”diqiuren”,”diqiu”,”di”,三种情况,余下的音节作为用户的后续输入.

4.排序,将所有的候选词按照优先级进行降序排列.

5.将所有的候选词绘制到UI上.

输入法与用户之间的交互是阻塞式的,用户每次输入后需要等待输入法返回结果.但是从用户体验(User Experience)的角度来说每次输入后都要等待输入法是相当不舒服的一种体验,因此我们需要将每次的响应时间缩减到用户可以忽略不计的程度(至少要在100ms以下,最好能在50ms以下).并且初始化(词典的构建)时间也不能太久,至少要在1s以下.这也是我们的输入法必须要采取高效的数据结构的原因,目前我们的初始化时间大概在500ms左右(这里说的是从选择输入法到弹出输入法界面的所有时间),用户每次输入的响应时间平均能达到50ms,偶尔有一些简拼的场景可能会超过100ms.

为了进一步提升用户体验,其中还采用了异步加载的方式.通过分析可以得知,虽然用户对输入法进行的操作是一个同步操作,输入法内核需要在得出结果前阻塞UI线程(实际上它们就是在同一个线程),但是当用户在连续输入时,其实他往往并不关心中间的结果.因此在这里就可以采用异步操作,当一个输入到来后,不阻塞UI线程,让用户可以继续输入,当下一个输入到来时,如果上一次的运算还没结束,就取消掉,直到一次运算完成后才将结果加载到UI上.通过这样的候选词异步加载,可以减少许多不必要的运算与UI操作,使用户明细感觉到流畅度的提升.

还有一些细节内容,主要涉及到用户体验,比如说中英文混输的处理,空格键,回车键的处理,以及退格键的行为等,这里就不具体展开了,因为和输入法核心部分关系不大.

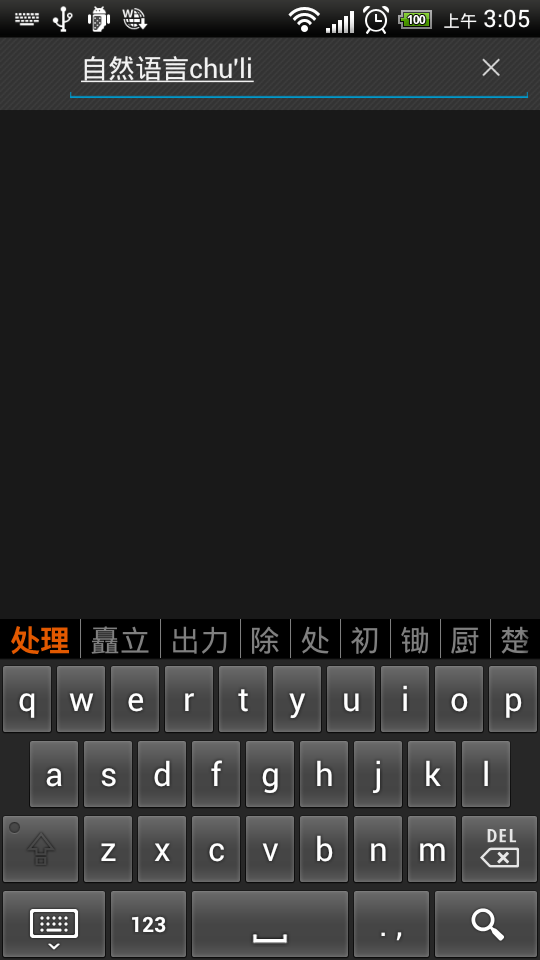


图3.输入法的实际运行画面

# 词库文件

词库，顾名思义，就是一个包含了大量词组，短句的文字库。它将拼音组合映射到中文词组，按照某种规则，通过输入法的其他模块，最终将符合的词组返回到用户界面，供用户使用。

在设计实现输入法的过程中，词库的设计是必须且重要的一个环节。词库的好坏直接影响着输入法的用户体验。一个好的词库，可以让用户在使用时更加得心应手，加快打字速度，并且拥有一个良好的工作心情。

在设计输入法的过程中，通过小组的内部讨论，我们结合我们所需要设计的输入法的需要，我们将词库的词条格式定义如下：

拼音1^' 拼音2^'…拼音n^' 词组or短句 词频

简单来说，词条的第一部分为拼音部分，每个音节间用符号“’”隔开，第二部分则为拼音部分所对应的中文词组（或者短句），第三部分则是这个词条的词频，即为一个权重值，值越大，说明该词在文章中出现的概率越高。在词条的不同部分间我们用制表符隔开。下面举个简单的例子：hai'shi'shan'meng 海誓山盟 1。

在各个不同的拼音间用符号“’”隔开，是为了防止出现歧义，例如xian与xi’an，如果没有符号进行标识，那么我们就无法快速识别这两者的区别。尽管有其它解决的方式，但在我们的设计中还是采用了这种解决方式。此外，词库中的词条是按照拼音的排序从a~z的顺序排列的，这样有利于后续工作的实现，减少查找的时间开销。

在我们的初始版本中，由于各种条件的限制，我们将各个单词的词频都初始化为1，因为我们无法获得一个较为准确的词频数据。在今后的版本中，我们会修改词频的初始值。我们的设想是通过对《人民日报》进行数据挖掘与分析，得到各个词出现的词频，并在我们的词库中加入新的单词。

在这次实验中，我们共设计了三个不同大小的词库，分别为8w，15w以及22w词汇规模的词库，以满足不同的需要。同时，在这些词库中，我们添加了现在流行的网络用语及新兴词汇，同时也添加了不少的唐诗、宋词等传统文学中的名言警句，来丰富我们的词库，期望得到良好的用户体验。

# 测试

不同于英文输入法中文拼音输入法在输入识别时面临着切音，分词，选词等诸多问题，而基于规律或是基于统计的算法都不能穷尽汉语的所有可能，尤其是在本项目中基于词法分析的算法，在面对规则性不强的新词、网络词汇很有可能导致出现在输入之后机器识别出的结果与人的预期结果有着不一致的地方，基于统计规律的由于训练集的有限性也存在着不尽如人意的地方。这使得黑箱测试尤为重要。

在本次试验测试中，采用单元检测的方法，依次检测切音，分词，选词等各个环节。

在切音环节我们随机的从训练集（词库共计有8万个词汇）中抽取1%，对其进行无差别切音检测，比较切音结果与预期是否相同。在比较结果中我们发现形如“xian（西安）”即第二个音无声母，会被识别成“xian（先）”而“先”这个词在日常生活中又是切实存在的，那么我们可以强制要求再输入“西安”时加入切音符“xi’an”，在接下来的工作中则会结合词频，上下文Ngrammar概率在“xi’an”识别的基础上加强对于“xian”基于统计规律的多重识别切音，智能识别成“xian（先）”或“xi’an（西安）”；形如“ganga（尴尬）”即的二个音的声母或起始音节被误识别为上一个音的尾音，即会被识别成“gang’a”这是我们所不能接受的，需要结合统计规律来避免这种情况。

在分词环节如键入“dajiazaoshanghao（大家早上好）”我们期望被识别为“dajia|zaoshang|hao”三个词，而由于训练集的有限性，汉语的多义性（歧义性）很有可能被识别成期望外的结果。比如“shuiguoranhoushu（水果然后熟）”我们希望分成“shuiguo|ranhou|shu”而不是“shui|guoran|hou|shu（水|果然|后熟）”，而这些问题只有通过大量的黑箱测试，找出问题句段，并用这些句段补充训练集才能更好的分词。

选词部分目前是基于词频的统计排序尚未结合上写文，目前检测主要是检测词频的排序以及词频改变是否能及时更新。

# 小组情况

本小组共有5个人.

Leader为周昊一(MF1233055),负责核心代码,以及用户界面的编写.

其余4个人为:

张绍文(MF123304),负责字典,词典的搜集,完善,以及后续工作中语言模型的训练.

郑健(MF1233054),同上.

张昊(MG1233046),同上.

周佳雨(MG1233091),负责测试用例的编写,并对程序进行测试.