

MIMIC 数据处理说明 1

1. 数据说明

MIMIC 数据由若干个表格构成，表格的每一行代表一个事件。

事件可能以下几种 ID 标识码：

Links to:

- PATIENTS on SUBJECT_ID
- ADMISSIONS on HADM_ID
- ICUSTAYS on ICUSTAY_ID

其中`subject_id`是病人 ID，`hadm_id`是病人的住院 ID，`icustay_id`表示病人在某次住院中进入 ICU 的 ID。

由于原始的 MIMIC 数据事情类型多，事件信息复杂，所以我们分别从行（减少事件类型和数量）和列的角度（减少事件信息）对表格的内容进行筛选。

2. 事件信息筛选

本步骤在数据库遍历每一个表格的全部内容，并从原始的表格中选取若干列并存储到本地。

选取的列内容可分为 4 个方面：

1. 所有的 ID 信息：包括`subject_id`、`hadm_id`、`icustay_id`
2. 事件的时间戳信息：一般表示事情的发生事件
3. 事件的类别信息：通过记录事件所在的表格+表示事件类型的 code 来唯一指定事件类型

4. 事件的其它特征：记录其它有价值的信息

19246_150542_283359	2126-07-22 09:00:00	inputevents_cv.30056	180.0 &
19246_150542_283359	2126-07-22 18:00:00	inputevents_cv.30056	120.0 &
19246_150542_283359	2126-07-22 20:00:00	inputevents_cv.30056	200.0 &
19246_150542_283359	2126-07-23 09:00:00	inputevents_cv.30056	120.0 &

上图就是表格 inputeventscv 表格抽取结果的示例。第一列分别表示了 \$subject\id、hadm_id、icustay_id\$ 3 个 ID，第二列是记录输入液体的时间，第四列是输入的总量和速率。

3.事件类型选择

我们统计了事件在病人中的覆盖率，并选取前 1000 种事件用于生成实验数据。其中覆盖率第 1000 的事件覆盖率在 5%左右。

admissions.admit	46520	(46520, 1.2678)	(46520, 0.8925)	(7537, 0.0021)	(7537, 0.0087)
diagnoses_icd	46520	(46520, 13.995)	(46520, 15.6701)	(0, 0)	(0, 0)
admissions.disch	46520	(46520, 1.2678)	(46520, 0.8925)	(7526, 0.0035)	(7526, 0.063)
icustays	46467	(46467, 2.648)	(46467, 1.9921)	(46467, 0.1313)	(46467, 2.7111)
labevents.51221	45383	(45383, 19.4312)	(45383, 29.5553)	(39792, 0.0586)	(39792, 0.3063)
labevents.51301	45335	(45335, 16.6163)	(45335, 25.998)	(39361, 0.0522)	(39361, 0.2944)
labevents.51265	45305	(45305, 17.1823)	(45305, 27.4339)	(39450, 0.0536)	(39450, 0.2958)
labevents.51222	45298	(45298, 16.6127)	(45298, 25.9671)	(39361, 0.0521)	(39361, 0.2941)
labevents.51248	45292	(45292, 16.5167)	(45292, 25.8417)	(39329, 0.0519)	(39329, 0.2941)
labevents.51249	45292	(45292, 16.52)	(45292, 25.8466)	(39330, 0.0519)	(39330, 0.2941)
labevents.51250	45292	(45292, 16.5163)	(45292, 25.8411)	(39329, 0.0519)	(39329, 0.2941)
labevents.51279	45292	(45292, 16.5168)	(45292, 25.8419)	(39329, 0.0519)	(39329, 0.2941)
labevents.51277	45286	(45286, 16.4929)	(45286, 25.8201)	(39319, 0.0519)	(39319, 0.2943)
patients.dob	44561	(44561, 1.0)	(44561, 0.0)	(0, 0)	(0, 0)
procedures_icd	42214	(42214, 5.6876)	(42214, 5.6217)	(0, 0)	(0, 0)
labevents.50971	41210	(41210, 20.5248)	(41210, 30.038)	(39505, 0.053)	(39505, 0.222)

体的事件类型统计信息在 sorted_stat.tsv 中，第一列表示事件类型，第二列表示事件在多少个病人中出现过。

4.事件特征处理

对于事件的特征，我们首先解析特征的数值类型，可能的数值类型有以下几种

- 1. 时间类型：表示事件的截止时间
- 2. 数值类型
- 3. 文本类型

如果一个事件的某个特征的任一种类型覆盖率超过 95%，则将这种类型作为这个特征的主要类型，这个特征的所有数据都按照这种类型进行解析。对于任一特征类型都不超过 95%的特征，我们将其抛弃。

对于数值特征，直接解析其数值即可。对于时间特征，将其与发生时间相减，转化为持续时间，当做数值特征处理。

对于文本特征，我们将其作为一种事件类型的细分类特征。如一个事件 A，它有一个文本特征 X，X 的取值可能有 20 中，那么就将事件 A 依据特征 X 转换为 20 种事件 $A_1, A_2, A_3, \dots, A_{20}$ 。为了防止事件种类爆炸，额外要求 X 的取值要少于 40 种。

生成的最终的 event 文件格式如下，每行 4 个值。分别是事件 ID（已经包含了文本特征）、*subject_id*、空格划分的数值特征（每种特征以下标：值的形式表示）、时间戳。

```
1990      441      379:7.0 2123-05-31 10:00:00
1990      18992    379:2.0 2124-05-01 10:00:00
1990      18822    379:0.0 2163-04-25 10:00:00
1990      20815    379:2.0 2147-06-05 10:00:00
1390      5725     195:0.12      2145-06-21 10:00:00
1990      12408    379:0.0 2187-04-23 10:00:00
1990      11976    379:2.0 2198-06-21 06:00:00
1990      4185     379:8.0 2177-05-22 06:00:00
1990      24366    379:2.0 2113-05-04 06:00:00
1990      4902     379:2.0 2184-07-08 06:00:00
1990      15886    379:3.0 2152-06-08 06:00:00
1990      3603     379:3.0 2123-06-17 06:00:00
1990      441      379:7.0 2123-06-03 06:00:00
1990      18992    379:15.8333330154      2124-05-04 06:00:00
1472      24366    199:139.200000763      2113-05-04 06:00:00
1472      1085     199:70.7999992371      2129-06-09 06:00:00
1472      4185     199:220.31999588      2177-05-22 06:00:00
1472      24877    199:287.999992371      2168-05-12 06:00:00
```

5.运行流程

1. 运行环境 python 库：python 2.7 numpy 1.11.1 PyGreSql 5.0.4；数据库：PostgreSQL 9.3.16；操作系统：Ubuntu 14.04
2. 参考 <https://mimic.physionet.org/tutorials/install-mimic-locally-ubuntu/> 将数据导入一个 Postgres 数据库中。默认将数据导入到模式 mimiciii 中。
3. 运行以下命令来做事件信息筛选，生成的数据会存储在当前文件下的 data 目录。

```
python extract.py --host 162.105.146.246 --user mimic --passwd mimic
```

其中--host 是数据库的 IP 地址，本地可用 localhost 代替；--user --passwd 分别表示数据库的用户和密码。可选参数--schema 表示 MIMIC 数据所在的模式，默认为 mimiciiii。

4. 运行以下命令，统计 data 目录下的事件的一些统计信息，来为后续的事件类型选择做准备，生成的数据将会存储于当前文件下的 stat 目录。

```
python stat_data.py
```

5. 运行 gather_stat.py 与 sort_stat.py 来整合之前的统计信息。生成的结果将会存储于 result 目录下。

```
python stat_value.py  
python gather_stat.py  
python sort_stat.py
```

6. 运行以下命令，进行事件类型选择，生成的 result/selected_features.tsv 存储了筛选之后的前 1000 个事件类型、事件特征以及事件的特征类型。生成的 result/feature_info.tsv 存储了事件和特征的具体含义。其中 gather_static_data.py 会从数据库抓取数据，需要数据库的相关参数。

```
python select_feature.py
```

```
python gather_static_data.py --host 162.105.146.246 --user mimic --passwd mi  
mic
```

7. 运行以下命令，进行事件的筛选。生成最终事件将会存储于 event 目录下。








```
python build_event.py
```
















```
python event_des.py
```

MIMIC 数据说明 2

6. 数据读入与处理

本步骤读入前一步生成的下列 tsv 文件，并输出初步生成好的事件序列。

-  admissions.admittime.tsv
-  admissions.deathtime.tsv
-  admissions.disctime.tsv
-  chartevents_1.tsv
-  chartevents_2.tsv
-  chartevents_3.tsv
-  chartevents_4.tsv
-  chartevents_5.tsv
-  chartevents_6.tsv
-  chartevents_7.tsv
-  chartevents_8.tsv
-  chartevents_9.tsv
-  chartevents_10.tsv
-  chartevents_11.tsv
-  chartevents_12.tsv
-  chartevents_13.tsv
-  chartevents_14.tsv

-  cptevents.tsv
-  datetimeevents.tsv
-  diagnoses_icd.tsv
-  feature_info.tsv
-  icustays.tsv
-  inputevents_cv.tsv
-  inputevents_mv.tsv
-  labevents.tsv
-  microbiologyevents.tsv
-  outputevents.tsv
-  patients.dob.tsv
-  patients.dod.tsv
-  prescriptions.tsv
-  procedureevents_mv.tsv
-  procedures_icd.tsv

- a) 读入上述 tsv 格式文件和 feature_info.tsv 文件，从 tsv 文件中抓取每个事件对应的患者，以及对应的事件类型，时间，事件的特征类型与特征值等信息存成一个字典。

- b) 以病人为关键字分类，并将事件序列按照时间顺序排序。
- c) 以病人顺序输出病人事件序列，输出格式为：起始一个数值 n （如图中的 45563），表示病人数量，之后数据分为 n 部分，对每个病人 i 的部分，起始是三个数值，分别为病人标号 p （图中第二行的 2），事件序列长度 w （图中第二行的 50），事件序列中选中为 label 的 labevent 事件的个数 l （图中第二行的 18）。之后 w 小部分每部分表示事件序列中的一个事件的描述（如 3-4 行为一个事件表示，5-7 行为另一个事件表示），事件描述分为两部分，第一部分是一行三个值（如图第三行），分别表示这个事件是不是 label 事件（是则为 1，否则为 0），事件的标号，事件发生的时间；若是 label 事件（第一个值为 1），则紧接下来一行会输出一个值（必定在 0, 1, 2 之间），表示这个 label 事件的类型，0 表示 normal, 1 表示 abnormal, 2 表示 delta。第二部分为 feature，第一行为 feature 数量 f ，后面 f 行为 feature 的标号和数量。详情见程序注释。（程序：extractor.cpp）

```
45563
2      50      18
0      4      2138-07-17 19:04:0
0
0      7      2138-07-17 21:20:0
1
1      2.20389
0      868      2138-07-17 21:48:0
1
85      0
0      314      2138-07-17 21:48:0
1
48      0
0      68      2138-07-17 21:48:0
1
27      0
0      70      2138-07-17 21:48:0
1
28      0
1      9      2138-07-17 21:48:0
1
1
2      0
1      18      2138-07-17 21:48:0
1
1
1
5      0
0      62      2138-07-17 21:48:0
1
24      0
1      21      2138-07-17 21:48:0
1
1
1
6      0
1      24      2138-07-17 21:48:0
1
1
7      0
1      27      2138-07-17 21:48:0
1
1
8      0
0      870      2138-07-17 21:48:0
1
86      0
0      64      2138-07-17 21:48:0
1
25      0
0      872      2138-07-17 21:48:0
```

7. 数据文件生成

本步骤读入前一步生成的病人事件序列文件 dataSeq.txt，并将其生成 numpyarray 的格式并输出到 h5py 格式的输出文件 Lab.h5 中。

选出事件序列中所有的 labtest 类型事件，对每个 labtest 事件，以其发生时间向前推半天的时间为终点，向前推 1000 个事件为起点（若不足 1000 个则起点为第一个事件），记录 label 为该 labtest 类型，labtest 的化验指

标（化验结果为正常【normal，标注为0】或异常【abnormal，标注为1】），labtest 的化验数值，起点坐标，终点坐标组成的五元组。

故一个病人的事件序列会在多个位置上分别对应多个 label，本结果为一序列多标签样例数据，最终输出到 h5py 格式的文件中储存。（参考程序 dataExt.py）

8. 数据集描述

Labtest 为一样例多 label 数据集。

打开数据集：f = h5py.File(‘Lab.h5’ , ‘r’)

查看内容：f.keys()

```
>>> f = h5py.File('Lab.h5')
>>> f.keys()
[u'event', u'feature', u'label', u'patient', u'time']
```

如图可见，有五张表

Patient:

```
>>> patient = f['patient']
>>> patient
<HDF5 dataset "patient": shape (45563,), type "<i8">
```

每个样例的 patient 为一个整数，表示病人编号，病人编号是唯一不重复的，每个病人编号唯一对应一个事件序列，一个事件序列，一组特征序列和一个标签序列。

Event:

```
>>> event = f['event']
>>> event.shape
(45563, 5000)
```

长度为 5000 的事件序列，不足 5000 后面 0 填充，范围在[0, 3418]，每个事件编号表示一个事件类型，示例如下。

```
>>> event = f['event']
>>> event[0]
array([ 4,  7, 868, ...,  0,  0,  0])
```

Time:

```
>>> time = f['time']
>>> time.shape
(45563, 5000)
```

长度为 5000 的时间序列，不足 5000 的后面空串填充，每个事件唯一对应一个时间。时间格式为 xxxx-xx-xx xx:xx:xx，示例如下：

```
>>> time[0]
array(['2138-07-17 19:04:0', '2138-07-17 21:20:0', '2138-07-17 21:48:0',
      ..., '', '', ''],
      dtype='<S18')
```

Label:

```
>>> label = f['label']
>>> label.shape
(45563, 3682, 5)
```

Label 表的维度为[样例个数, 3682, 5]，每行对应一个 label 列表，表示该行的事件序列和时间序列唯一对应的 label 列表。长度为 3682，每个 label 为一个五元组。

```
>>> label[0]
array([[ 9.,  0.,  1.,  5.,  0.],
       [ 18.,  0.,  1.,  6.,  0.],
       [ 21.,  0.,  1.,  8.,  0.],
       ...,
       [  0.,  0.,  0.,  0.,  0.],
       [  0.,  0.,  0.,  0.,  0.],
       [  0.,  0.,  0.,  0.,  0.]])
```

如上图，每个 label 序列由若干个 label 组成，每个 label 为一个五元组，分别表示 label 的 labtest 类型编号，labtest 数值，labtest 分类指标【0-normal, 1-abnormal, 2-delta】，labtest 预测序列的结束位置，labtest 预测序列的开始位置。

如第一行表示一个 label，其对应的 lab 事件为 9 号事件，化验数值为 0，验证类型为 1-abnormal，这个 label 的预测对应着从 0 到 5 这段序列，也即我们以后将用事件和时间序列中 0-5 这部分之间的信息来预测这个 label 的相关信息。

Feature:

```
>>> feature = f['feature']
>>> feature
<HDF5 dataset "feature": shape (45563, 5000, 6), type "<f8">
```

feature 表为一个大小为[5000, 6]的数组，每个事件对应一个六维的特征值，其中 0, 2, 4 行为特征标号，1, 3, 5 行分别为 0, 2, 4 对应特征的特征值。特征可能有 0 到 3 个，若不足 6 维则补零对齐。

如下图中，第一个事件没有特征，第二个事件有一个特征，其值为 2.20389。


```
>>> feature[0]
array([[ 0.      ,  0.      ,  0.      ,  0.      ,  0.      ,  0.      ],
       [ 1.      , 2.20389,  0.      ,  0.      ,  0.      ,  0.      ],
       [ 85.      ,  0.      ,  0.      ,  0.      ,  0.      ,  0.      ],
       ...,
       [ 0.      ,  0.      ,  0.      ,  0.      ,  0.      ,  0.      ],
       [ 0.      ,  0.      ,  0.      ,  0.      ,  0.      ,  0.      ],
       [ 0.      ,  0.      ,  0.      ,  0.      ,  0.      ,  0.      ]])
```

9. 数据切分

实验代码最终使用的是一序列一标签的样例数据，故而最后还需要对数据进行选择和切分。

数据选择即为选定 label 中事件类型为某一特定类型的一组数据（因为原数据类型划分时，统一化验的正常和非正常会被分为两种事件类型，故而这里的一组也就是两个）

数据切分十分简单，读入之前结果的 h5py 文件，选择一个化验事件作为 label，并按照 label 中的起点和终点将原医疗事件序列截断，起点和终点之间的序列对应此 label 生成一个新的样例数据，并以化验指标为新标签，以此处理完成则产生了新的一序列一标签数据，也即为我们实验运行所使用的数据。（参考代码：lab_process_data.py）

10. 切分后的数据格式

切分后数据发生了一些变化，我们以 train.h5，切分为长度为 200（这个长度为参数，可以在 9 中对应程序的参数列中修改），选取化验血液钾离子浓度的医疗事件为标签（这个也为参数，可以在 9 中对应程序处修改）的事件序列为例来讲解数据组织格式。

```
f = h5py.File('train.h5')
```

```
f.keys()
```

可以看到切分后数据中有五张表：

```
>>> f = h5py.File('train.h5')
>>> f.keys()
[u'events', u'feature_id', u'feature_value', u'labels', u'times']
```

Events:

```
>>> event = f['events']
>>> event
<HDF5 dataset "events": shape (535419, 200), type "<i2">
```

每个样例有一个事件序列，长度为 200，取值范围为[0,3418]，末尾 0 填充。

Time:

```
>>> time = f['times']
>>> time
<HDF5 dataset "times": shape (535419, 200), type "<f4">
```

每个事件序列唯一对应一个时间序列，长度同为 200，每个事件唯一对应一个时间，末尾空串填充。

Feature_id 和 feature_value:

```
>>> fid = f['feature_id']
>>> fvalue = f['feature_value']
>>> fid
<HDF5 dataset "feature_id": shape (535419, 200, 3), type "<i2">
>>> fvalue
<HDF5 dataset "feature_value": shape (535419, 200, 3), type "<f4">
```

每个事件唯一对应一组 feature_id 和 feature_value，将原本的 6 维数组拆成两个 3 维数组，feature_id 中保存原 feature 中的特征标号(0,2,4 位置的值)，feature_value 中保存原 feature 中的特征值(1,3,5 位置的值)。

Label:

```
>>> label = f['labels']
>>> label
<HDF5 dataset "labels": shape (535419,), type "<f4">
```

每个事件序列对应一个 label，label 取值只有 0 和 1，分别表示钾离子浓度测试正常和钾离子浓度测试异常。

11. 运行流程

- 运行环境 python 库: python 2.7 numpy 1.11.1 h5py 2.6.0; 运行 g++ 版本: gcc version 4.8.4 操作系统: Ubuntu 14.04
- 编译数据处理程序: g++ extractor.cpp -o extractor, 运行 ./extractor。该程序读入以下 tsv 文件:



admissions.admittime.tsv
admissions.deathtime.tsv
admissions.dischtime.tsv
chartevents_1.tsv
chartevents_2.tsv
chartevents_3.tsv
chartevents_4.tsv
chartevents_5.tsv
chartevents_6.tsv
chartevents_7.tsv
chartevents_8.tsv
chartevents_9.tsv
chartevents_10.tsv
chartevents_11.tsv
chartevents_12.tsv
chartevents_13.tsv
chartevents_14.tsv
cptevents.tsv
datetimeevents.tsv
diagnoses_icd.tsv
feature_info.tsv
icustays.tsv
inputevents_cv.tsv
inputevents_mv.tsv
labevents.tsv
microbiologyevents.tsv
outputevents.tsv
patients.dob.tsv
patients.dod.tsv
prescriptions.tsv
procedureevents_mv.tsv
procedures_icd.tsv

并读入 feature_info.tsv 文件，故而请将这些文件放在 extractor 的同一目录下。

本步骤生成一个文件 dataSeq.txt，为初步生成的事件序列文件。

- c) 运行数据文件生成程序：python dataExt.py。该程序读入 dataSeq.txt 文件和 feature_info.tsv 文件，并输出一个文件 Lab.h5 为最终的数据文件。

- d) 运行数据切分程序：`python lab_process_data.py`。该文件读入 `Lab.h5` 文件，并输出三个文件 `train.h5`，`valid.h5`，`test.h5`，分别表示切分好的数据集，三者依次为训练中使用的训练集，交叉验证集和测试集。