语音识别：从入门到精通

第5章 GMM-HMM作业代码介绍

主讲人 龙岩

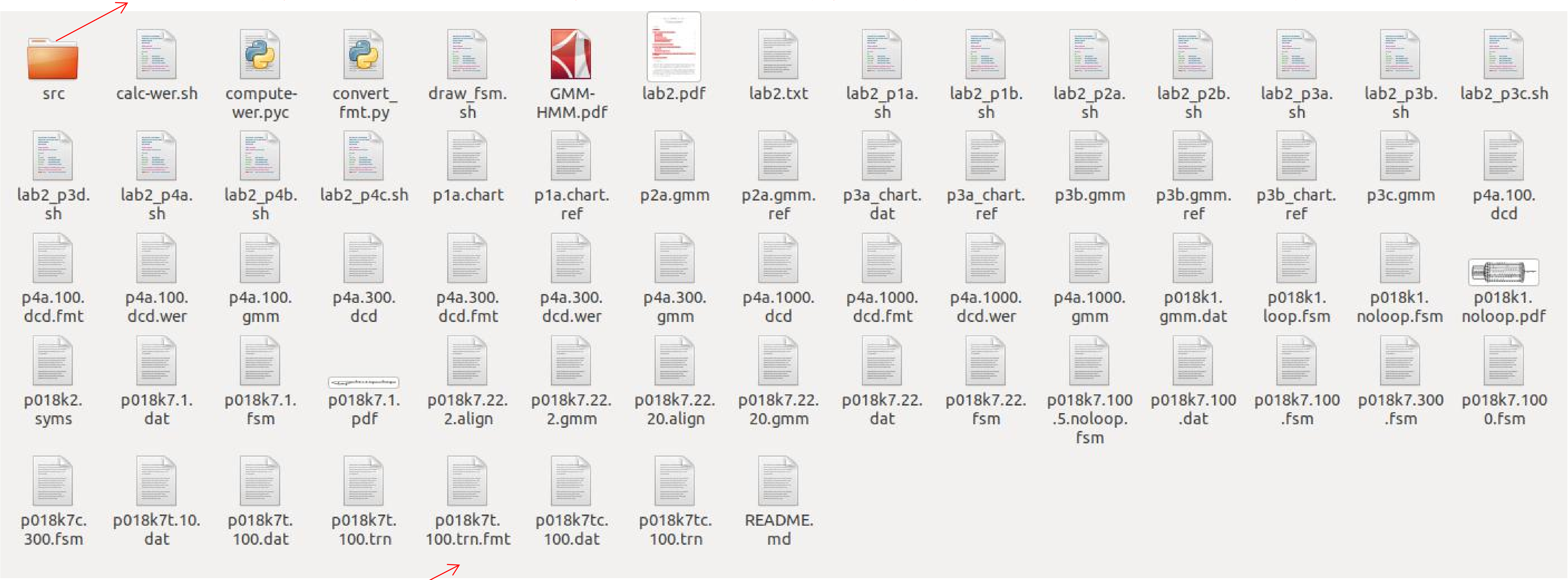
算法工程师
第一期优秀学员

作业

- 作业地址： https://github.com/nwpuaslp/ASR_Course/tree/master/05-GMM-HMM/
- 作业内容：
  - Viterbi解码
  - 估计GMM参数
  - 前向后向训练，利用前向后向训练估计GMM参数
- 作业中的几个重要文件：
  - README.md: 如何安装、编译、填写代码、对比结果。
  - lab2.pdf: 原始作业说明，需要细读。
  - src: src目录下为源代码文件。
  - lab2.txt: 提示思考的几个问题。

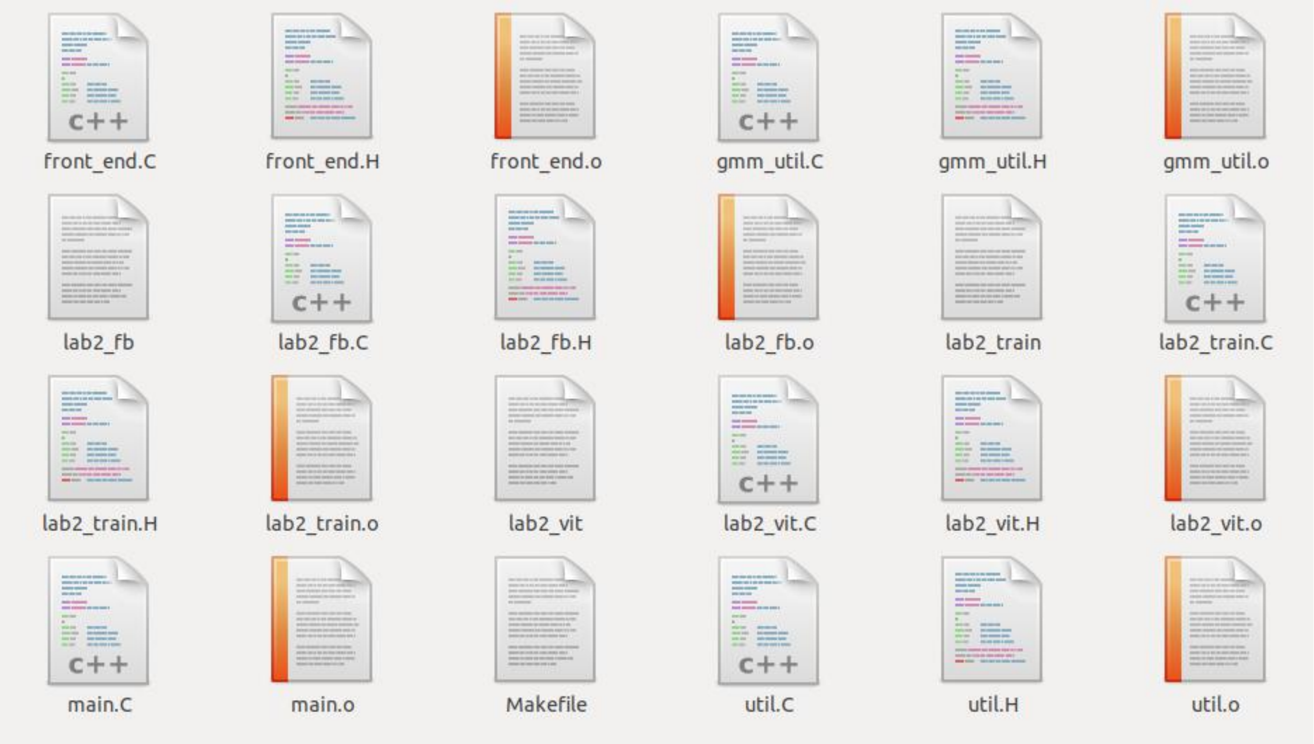
总体文件结构：

src/ 为源代码文件夹，其中包括本节课用到的所有代码文件，其中有五个位置需要我们完成



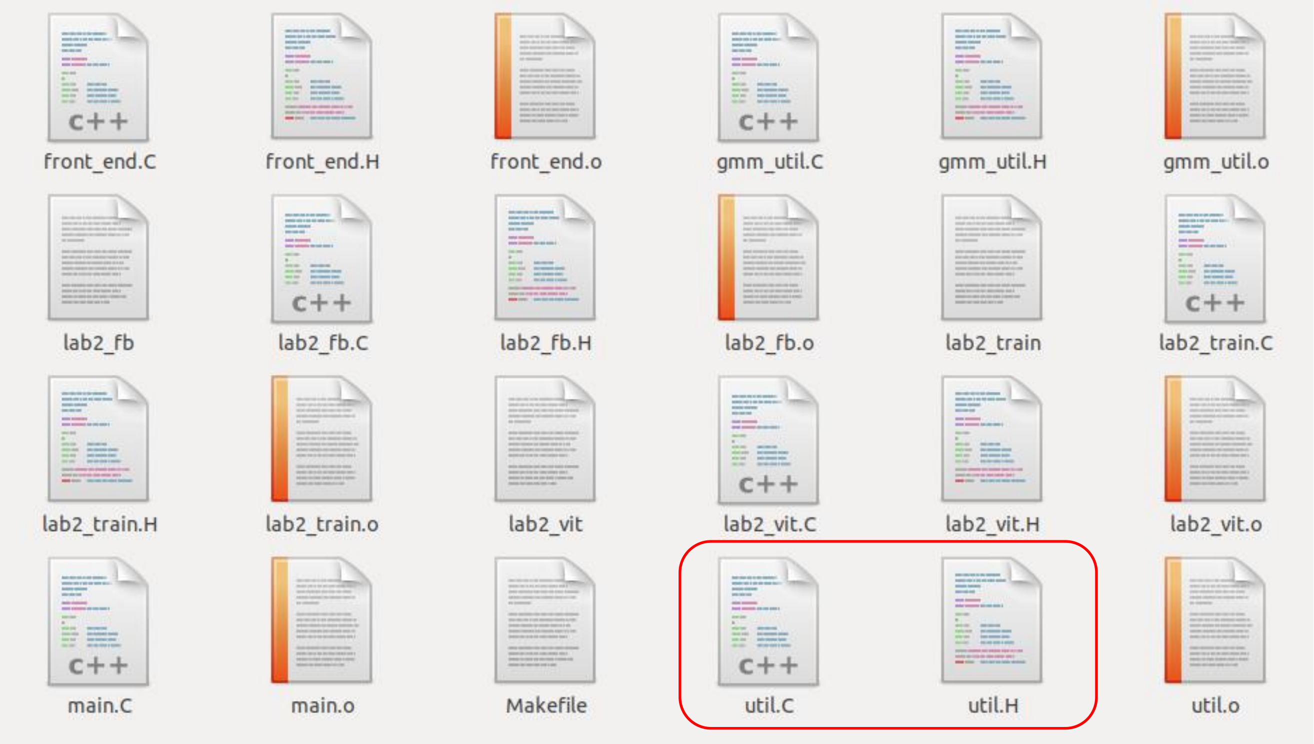
1.帮助我们运行作业代码的脚本文件sh；
2.存储数据、模型参数的文件如:gmm,fsm,dat等；
3.README.md

文件夹 src/



完成代码：
1. lab2_vit.c 中一处代码
2. gmm_util.c 中两处代码
3. lab2_fb.c 中两处代码

文件夹 src/



其中主要的类和实现代码在util.H和util.C中

README.md

基于GMM-HMM的语音识别系统

要求和注意事项

1. 认真读lab2.pdf, 思考lab2.txt中的问题
2. 理解数据文件
3. ref文件作为参考输出，用diff命令检查自己的实现得到的输出和ref是否完全一致
4. 实验中实际用的GMM其实都是单高斯
5. 阅读util.h里面的注释，Graph的注释有如何遍历graph中state上所有的arc的方法。
6. 完成代码
    - lab2_vit.C中一处代码
    - gmm_util.C中两处代码
    - lab2_fb.C中两处代码

README.md

作业说明

安装

该作业依赖g++, boost库和make命令，按如下方式安装:

- MAC: brew install boost (MAC下g++/make已内置）
- Linux(Ubuntu): sudo apt-get install make g++ libboost-all-dev
- Windows: 请自行查阅如何安装作业环境。

编译

对以下三个问题，均使用该方法编译。

```
make -C src
```

注意运行位置(路径)

作业

```
    throw runtime_error("GMM doesn't have single component.");
int gaussIdx = m_gmmSet.get_gaussian_index(gmmIdx, 0);
int dimCnt = m_gmmSet.get_dim_count();

//  BEGIN_LAB
//
//  Input:
//      "dimCnt" holds the dimension of the Gaussian and the
//      acoustic feature vector.
//      The acoustic feature vector is held in
//      "feats[0 .. (dimCnt-1)]".
//      "gaussIdx" is the index of the Gaussian to be updated.
//      "posterior" is the posterior count of this Gaussian for
//      the current frame.
//
//      The values of the current means and variances can be
//      accessed via the object "m_gmmSet".
//
//  Output:
//      You should update the counts stored in
//
//      m_gaussCounts[0 .. (#gaussians-1)]
//      m_gaussStats1(0 .. (#gaussians-1), 0 .. (dimCnt - 1))
//      m_gaussStats2(0 .. (#gaussians-1), 0 .. (dimCnt - 1))
//
//      "m_gaussCounts" is intended to hold the total occupancy count
//      of each Gaussian; "m_gaussStats1" is intended for
//      storing some sort of first-order statistic for each
//      dimension of each Gaussian; and "m_gaussStats2" is intended for
//      storing some sort of second-order statistic for each
//      dimension of each Gaussian.  The statistics you take
//      need to be sufficient for doing the reestimation step below.
//
//      These counts have all been initialized to zero
//      somewhere else at the appropriate time.

// suppose each GMM only has one component

//  END_LAB
//
```

在这里填上你的代码

p1

- 内容：完成lab2_vit.C中的用viterbi解码代码.

主要练习Viterbi解码

- 运行:
  - ./lab2_p1a.sh
  - ./lab2_p1b.sh
- 比较结果: 比较你的程序运行结果p1a.chart和参考结果p1a.chart.ref，可以使用vimdiff p1a.chart p1a.chart.ref进行比较，浮点数值差在一定范围内即可。

我们的结果输出是p1a.chart，同给出的p1a.chart.ref的值进行比较，来判断代码是否正确运行。

```
#!/bin/bash -e

if [[ -e ./src/lab2_vit ]] ; then
    binStr="./src/lab2_vit"
else
    echo "Couldn't find program to execute."
    exit 1
fi

$binStr --gmm p018k7.22.20.gmm --audio_file p018k7.1.dat \
    --graph_file p018k1.noloop.fsm --word_syms p018k2.syms \
    --dcd_file /dev/null --chart_file p1a.chart
```

执行编译好的可执行文件lab2_vit，并传入参数(数据，模型，输出文件)

p1.维特比解码

p018k7.1.dat

```
% name: utt2a
% type: matrix
% rows: 14080
% columns: 1
 3
 1
 3
 2
 2
 3
 2
 1
 1
 0
 0
 2
 -2
 -2
 0
 -1
 0
 -1
 -2
 1
 0
 -3
 0
 -4
 -2
```

音频输入数据

```bash
#!/bin/bash -e

if [[ -e ./src/lab2_vit ]] ; then
    binStr="./src/lab2_vit"
else
    echo "Couldn't find program to execute."
    exit 1
fi


$binStr --gmm p018k7.22.20.gmm --audio_file p018k7.1.dat \
    --graph_file p018k1.noloop.fsm --word_syms p018k2.syms \
    --dcd_file /dev/null --chart_file p1a.chart
```

p1.维特比解码

p1a.chart

```
% name: utt2a_probs
% type: matrix
% rows: 69
% columns: 123
 -1.70141e+38 0 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e
 -1.70141e+38 -1.70141e+38 -27.1463 -8.34394 -12.9549 -14.8215 -20.1241 -24.2817 -12.5074
 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+
 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+38 -1.70141e+
 -1.70141e+38 -1.70141e+38 -28.6783 -31.1717 -31.7819 -31.5971 -37.6664 -43.162 -38.9913 -
 -1.70141e+38 -1.70141e+38 -29.8439 -31.1844 -31.865 -31.6497 -37.8282 -43.4865 -36.276 -
 -1.70141e+38 -1.70141e+38 -31.0862 -32.5912 -33.0037 -32.0553 -39.2603 -46.9203 -37.5446
 -1.70141e+38 -1.70141e+38 -30.0749 -31.769 -32.8782 -32.2916 -38.4424 -47.4021 -36.5717 -
 -1.70141e+38 -1.70141e+38 -41.4395 -36.1584 -37.3397 -38.4891 -43.6257 -56.3763 -40.5883
 -1.70141e+38 -1.70141e+38 -38.8824 -38.3059 -39.5831 -38.9003 -45.0086 -56.7307 -44.142 -
 -1.70141e+38 -1.70141e+38 -49.1881 -40.7919 -43.5146 -43.3165 -49.6803 -58.7558 -46.2293
 -1.70141e+38 -1.70141e+38 -52.6021 -43.5626 -45.7069 -46.5464 -53.4178 -56.1913 -48.1471
 -1.70141e+38 -1.70141e+38 -41.1819 -42.2621 -43.0297 -42.9531 -47.2551 -53.8706 -43.9179
 -1.70141e+38 -1.70141e+38 -39.044 -42.2061 -42.5505 -42.7309 -45.9554 -60.9933 -46.3227 -
 -1.70141e+38 -1.70141e+38 -44.8157 -44.6847 -46.8749 -44.4223 -49.7103 -65.9763 -49.609 -
 -1.70141e+38 -1.70141e+38 -50.1223 -50.0377 -52.8364 -51.9509 -57.7592 -73.5919 -63.9746
```

维特比解码的过程值

```
% name: utt2a_arcs
% type: matrix
% rows: 69
% columns: 123
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -
 -1 -1 5 9 8 2 1 7 6 0 3 4 10 11 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 12 13 14 15 16 17 18 19 20 2
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 36 37 40 41 44 45 48 49 52 5
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 36 37 40 41 44 45
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 40 41 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 12 13 14 15 16 17
 -1 -1 114 118 117 111 110 116 115 109 112 113 119 -1 36 37 14 15 16 17
```

arc弧对应的id

弧？

p1.维特比解码

p018k1.noloop.fsm

```
# states: 122
# input-voc: /dynamic/
# output-voc: /u/stanchen/lma/018/p018/../c018/c018m1.wdsp
1	9	0	EIGHT
1	6	6	FIVE
1	5	15	FOUR
1	10	24	NINE
1	11	33	OH
1	2	36	ONE
1	8	45	SEVEN
1	7	60	SIX
1	4	72	THREE
1	3	81	TWO
1	12	87	ZERO
1	13	99	~SIL
2	14	36	<epsilon>
2	15	37	<epsilon>
3	16	81	<epsilon>
3	17	82	<epsilon>
4	18	72	<epsilon>
```

行数 state state gmm word

弧 arc
对应的类在util.H中，
class Arc

```
115 118 56	<epsilon>
116 116 97	<epsilon>
116 47	98	<epsilon>
116 119 98	<epsilon>
117 47	71	<epsilon>
117 117 71	<epsilon>
118 118 56	<epsilon>
118 120 57	<epsilon>
119 47	98	<epsilon>
119 119 98	<epsilon>
120 120 57	<epsilon>
120 121 58	<epsilon>
121 121 58	<epsilon>
121 47	59	<epsilon>
121 122 59	<epsilon>
122 47	59	<epsilon>
122 122 59	<epsilon>
47
84
```

p1.维特比解码

util.H

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Arc class.
 *
 *   Holds a single arc in a graph/FSM.  With each graph, there is
 *   an implicitly associated GmmSet and an explicitly associated
 *   SymbolTable (see Graph::get_word_sym_table()).
 *   An arc holds a destination state;
 *   an optional GMM index (corresponding to a GMM in the GmmSet);
 *   an optional word index (corresponding to a word in the SymbolTable);
 *   and a log prob (base e).
 *   Source state information is not present; if you have the arc ID
 *   (see Graph::get_first_arc_id(), Graph::get_arc()), you can look this
 *   up using Graph::get_src_state().
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class Arc {
 public:
  /** Ctor; initializes fields to default values. **/
  Arc() : m_dst(0), m_gmmIdx(-1), m_wordIdx(0), m_logProb(0.0) {}
```

```
private:
 /** Destination state. **/
 unsigned m_dst;
```

目标状态

```
 /** GMM index, or -1 if not present. **/
 int m_gmmIdx;
```

arc的gmm id

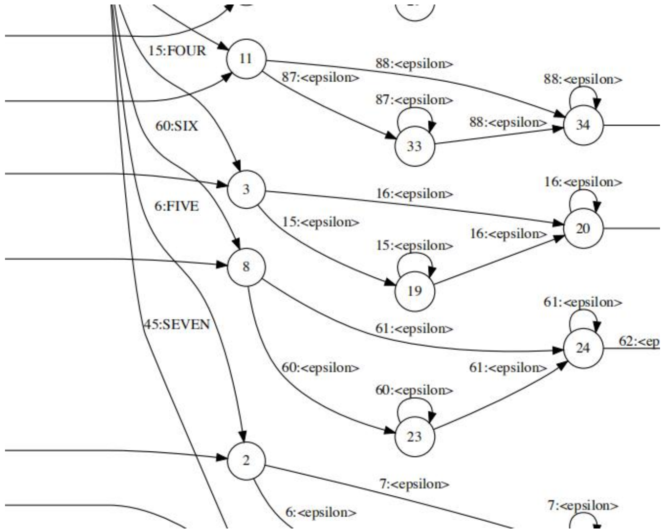
```
 /** Word index, or 0 if not present/epsilon. **/
 unsigned m_wordIdx;
```

arc对应的word

```
 /** Log prob base e. **/
 float m_logProb;
```

跳转概率

p018k1.noloop.pdf



p1.维特比解码

util.H:
class Graph，加载了fsm文件中数据，保存了Arc列表，并提供了很多接口，如获取初始state，获取arc等。

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Graph/FSM class.
 *
 *   This object holds a graph as needed for training or decoding.
 *   A graph has a set of states numbered starting from 0
 *   (see #get_state_count()), one of
 *   which is a start state (see #get_start_state()),
 *   and many of which may be final states (see #is_final_state(),
 *   #get_final_state_list()).  Final states have an associated final
 *   log prob (see #get_final_log_prob()).
 *
 *   For each state, you can access the list of outgoing arcs using
 *   #get_arc_count(), #get_first_arc_id(), and #get_arc().
 *   Each arc has an associated ID, which is used for iterating
 *   through arcs.  Here is an example of how to iterate through
 *   the outgoing arcs of state @p stateIdx of graph @p graph.
 *
 *   @code
 *   //  Get number of outgoing arcs.
 *   int arcCnt = graph.get_arc_count(stateIdx);
 *   //  Get arc ID of first outgoing arc.
 *   int arcId = graph.get_first_arc_id(stateIdx);
 *   for (int arcIdx = 0; arcIdx < arcCnt; ++arcIdx)
 *       {
 *       Arc arc;
 *       //  Place arc with ID "arcId" in "arc"; set "arcId"
 *       //  to arc ID of the next outgoing arc.
 *       arcId = graph.get_arc(arcId, arc);
 *
 *       //  You can now access elements of the Arc "arc", e.g.,
 *       int dstState = arc.get_dst_state();
 *       ...
 *       }
 *   @endcode
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class Graph {
 public:
  /** Ctor; loads from file @p fileName if argument present,
   *   and loads sym table from file @p symFile if argument present.
   **/
```

p1.维特比解码

p018k7.22.20.gmm

gmm模型的μ和对角矩阵的值

```
% name: gaussParams
% type: matrix
% rows: 102
% columns: 24
1.32639 6.60004 -1.26327 0.689253 1.1982 0.864616 0.478486 1.83185 -3.0023 0.77282 -0.500155
4.16635 2.04791 -0.0861596 0.976782 3.42422 0.367643 2.05445 1.09355 -3.29116 0.105375 0.159
-0.274429 1.74715 2.17292 0.351595 3.78784 0.250512 -0.852468 0.94553 -3.16182 0.301602 -0.6
1.96705 3.74624 2.90712 0.540845 2.34491 0.651479 1.94168 0.17576 -0.788577 0.40172 0.254892
-0.355743 2.27778 2.62647 0.408097 1.42787 0.136553 1.39812 0.0213317 0.24192 0.153607 0.254
-0.19848 4.07069 0.688211 1.45145 1.13602 0.348168 0.401133 0.602149 -0.0614162 0.202511 -0.
-2.19244 2.55908 -0.245306 0.57884 -0.361802 0.409575 0.916781 0.234671 -0.0949448 0.267796
1.60406 6.9397 0.0368008 0.372589 -0.661129 0.117898 0.282947 0.0165914 -0.319023 0.0724419
6.97766 0.481943 -2.01084 0.161006 -0.604437 0.260016 0.133259 0.671883 -1.54701 0.191875 1.
4.40298 0.137085 -4.26215 0.380116 -1.42316 0.095684 -0.855769 0.252391 -0.593966 0.0875204
5.04996 0.132081 -2.13517 0.259231 -0.446328 0.100116 1.00464 0.0707547 -1.19159 0.0396016 0
4.24592 0.324861 -2.34525 1.11582 -0.0889705 0.171049 -0.303064 0.358359 -2.27305 0.260015 -
4.35632 3.50617 0.528146 0.901434 0.495062 0.427698 1.79783 0.774893 -0.91724 0.0741842 -0.7
-1.60305 5.31535 -0.655801 1.64305 0.258832 0.712575 1.17277 0.321057 0.10183 0.107571 0.130
```

util.H中

```
/** Returns mean for dimension @p dimIdx for Gaussian  with index
 *   @p gaussIdx.
 *   Gaussians and dimensions are numbered starting from 0.
 **/
double get_gaussian_mean(unsigned gaussIdx, unsigned dimIdx) const {
  assert((gaussIdx < m_gaussParams.size1()) &&
         (2 * dimIdx < m_gaussParams.size2()));
  return m_gaussParams(gaussIdx, 2 * dimIdx);
}

/** Returns variance for dimension @p dimIdx for Gaussian with index
 *   @p gaussIdx.
 *   Gaussians and dimensions are numbered starting from 0.
 **/
double get_gaussian_var(unsigned gaussIdx, unsigned dimIdx) const {
  assert((gaussIdx < m_gaussParams.size1()) &&
         (2 * dimIdx + 1 < m_gaussParams.size2()));
  return m_gaussParams(gaussIdx, 2 * dimIdx + 1);
}
```

```
/** For each Gaussian, alternating mean + var for each dim. **/
matrix<double> m_gaussParams;
```

p1.维特比解码

1.我们有输入音频数据p018k7.1.dat，并且代码提供了特征处理；
2.我们有了.gmm文件中gmm的参数，可以计算高斯分布；
3.我们有.fsm文件中，以“弧arc”形式，提供了状态转移概率，状态与gmm关系，弧对应的word；

所以，我们有了GMM-HMM模型的所有参数，可以进行解码任务

p1.维特比解码

lab2_vit.H

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Cell in dynamic programming chart for Viterbi algorithm.
 *
 *   Holds Viterbi log prob; and arc ID of best incoming arc for backtrace.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class VitCell {
 public:
  /** Ctor; inits log prob to g_zeroLogProb and arc ID to -1. **/
  VitCell() : m_logProb(g_zeroLogProb), m_arcId(-1) {}

#ifndef SWIG
#ifndef DOXYGEN
  //  Hack; for bug in matrix<> class in boost 1.32.
  explicit VitCell(int) : m_logProb(g_zeroLogProb), m_arcId(-1) {}
#endif
#endif

  /** Sets associated log prob and arc ID. **/
  void assign(double logProb, int arcId) {
    m_logProb = logProb;
    m_arcId = arcId;
  }

  /** Returns log prob of cell. **/
  double get_log_prob() const { return m_logProb; }

  /** Returns arc ID of cell. **/
  int get_arc_id() const { return m_arcId; }

 private:
  /** Forward Viterbi logprob. **/
  double m_logProb;

  /** ID of best incoming arc, for traceback. **/
  int m_arcId;
};
```

class VitCell:
用于动态存储chart单元，
保存viterbi解码的数值
矩阵

p1.维特比解码

lab2_vit.H

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Encapsulation of main loop for Viterbi decoding.
 *
 *   Holds global variables and has routines for initializing variables
 *   and updating them for each utterance.
 *   We do this so that we can call this code from Java as well.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class Lab2VitMain {
 public:
  /** Initialize all data given parameters. **/
  Lab2VitMain(const map<string, string>& params);

  /** Called at the beginning of processing each utterance.
   *   Returns whether at EOF.
   **/
  bool init_utt();

  /** Called at the end of processing each utterance. **/
  void finish_utt(double logProb);

  /** Called at end of program. **/
  void finish();

  /** Returns decoding graph/HMM. **/
  const Graph& get_graph() const { return m_graph; }

  /** Returns matrix of GMM log probs for each frame. **/
  const matrix<double>& get_gmm_probs() const { return m_gmmProbs; }

  /** Returns DP chart. **/
  matrix<VitCell>& get_chart() { return m_chart; }

  /** Returns vector to place decoded labels in. **/
  vector<int>& get_label_list() { return m_labelList; }
```

class Lab2VitMain:
作业维特比解码部分的主要类，大家可以简单浏览下它的属性，如果时间充沛，可以理解它的设计思路。

p1.维特比解码

lab2_vit.C

最下面，程序执行的入口，main_loop

```bash
#!/bin/bash -e

if [[ -e ./src/lab2_vit ]] ; then
    binStr="./src/lab2_vit"
else
    echo "Couldn't find program to execute."
    exit 1
fi


$binStr --gmm p018k7.22.20.gmm --audio_file p018k7.1.dat \
    --graph_file p018k1.noloop.fsm --word_syms p018k2.syms \
    --dcd_file /dev/null --chart_file p1a.chart
```

```cpp
void main_loop(const char** argv) {
  map<string, string> params;
  process_cmd_line(argv, params);

  Lab2VitMain mainObj(params);
  while (mainObj.init_utt()) {
    double logProb = viterbi(mainObj.get_graph(), mainObj.get_gmm_probs(),
                             mainObj.get_chart(), mainObj.get_label_list(),
                             mainObj.get_acous_wgt(), mainObj.do_align());
    mainObj.finish_utt(logProb);
  }
  mainObj.finish();
}
```

接受参数，并计算mfcc，gmm_probs等

维特比算法实现，我们需要写代码的位置

p1.维特比解码

lab2_vit.C

```
bool Lab2VitMain::init_utt() {
  if (m_audioStrm.peek() == EOF) return false;

  m_idStr = read_float_matrix(m_audioStrm, m_inAudio);
  cout << "Processing utterance ID: " << m_idStr << endl;
  m_frontEnd.get_feats(m_inAudio, m_feats);
  if (m_feats.size2() != m_gmmSet.get_dim_count())
    throw runtime_error("Mismatch in GMM and feat dim.");
  if (m_doAlign) {
    if (m_graphStrm.peek() == EOF)
      throw runtime_error(
          "Mismatch in number of audio files "
          "and FSM's.");
    m_graph.read(m_graphStrm, m_idStr);
  }
  if (m_graph.get_gmm_count() > m_gmmSet.get_gmm_count())
    throw runtime_error(
        "Mismatch in number of GMM's between "
        "FSM and GmmSet.");
  m_gmmSet.calc_gmm_probs(m_feats, m_gmmProbs);
```

⋮

大家完成作业后也可以深入去追溯下函数内的实现

p1.维特比解码

lab2_vit.C

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Routine for Viterbi decoding.
 *
 *   @param graph HMM/graph to operate on.
 *   @param gmmProbs Matrix of log prob for each GMM for each frame.
 *   @param chart Dynamic programming chart to fill in; already
 *       allocated to be of correct size and initialized with default values.
 *   @param outLabelList Indices of decoded output tokens are placed here.
 *   @param acousWgt Acoustic weight.
 *   @param doAlign If true, return GMM indices rather than word indices
 *       in @p outLabelList.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
double viterbi(const Graph& graph, const matrix<double>& gmmProbs,
               matrix<VitCell>& chart, vector<int>& outLabelList,
               double acousWgt, bool doAlign) {
```

p1.维特比解码

lab2_vit.C

```
  //  The code for calculating the final probability and
  //  the best path is provided for you.
  return viterbi_backtrace(graph, chart, outLabelList, doAlign);
}

/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Routine for Viterbi backtrace.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
double viterbi_backtrace(const Graph& graph, matrix<VitCell>& chart,
                         vector<int>& outLabelList, bool doAlign) {
  int frmCnt = chart.size1() - 1;
  int stateCnt = chart.size2();
```

当我们计算完chart后，可以利用viterbi_backtrace()来获取解码结果

p2

- 内容：估计模型参数,不使用前向后向算法计算统计量，而是用viterbi解码得到的最优的一条序列来计算统计量，叫做viterbi-EM.给定align（viterbi解码的最优状态(或边）序列)，原始语音和GMM的初始值，更新GMM参数。完成src/gmm_util.C中两处代码。
- 运行：./lab2_p2a.sh
- 比较结果： 如p1，比较p2a.gmm p2a.gmm.ref

```
#!/bin/bash -e

if [[ -e ./src/lab2_train ]] ; then
    binStr="./src/lab2_train"
else
    echo "Couldn't find program to execute."
    exit 1
fi

$binStr --audio_file p018k7.22.dat --align_file p018k7.22.20.align \
    --iters 1 --in_gmm p018k1.gmm.dat --out_gmm p2a.gmm
```

p2.GMM-EM估计

```bash
#!/bin/bash -e

if [[ -e ./src/lab2_train ]] ; then
    binStr="./src/lab2_train"
else
    echo "Couldn't find program to execute."
    exit 1
fi

$binStr --audio_file p018k7.22.dat --align_file p018k7.22.20.align \
    --iters 1 --in_gmm p018k1.gmm.dat --out_gmm p2a.gmm
```

音频数据，注意数量上与p1中不同

对齐文件，标记每个frame对应哪个gmm

任务p2中，我们给了对齐文件，相当于给了后验概率，那么我们就可以统计特征frame，用EM算法来更新GMM的参数值。

p2.GMM-EM估计

gmm_util.H:
GmmStats统计、更新GMM时用的类，用来保存中间统计值，并且有一个GmmSet属性

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   GMM statistics for reestimating a GmmSet.
 *
 *   This object stores and updates the statistics required for reestimating
 *   a set of GMM's.  The method #reestimate() reestimates the
 *   associated GmmSet.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class GmmStats {
 public:
  /** Ctor; @p gmmSet is the object to reestimate and @p params
   *   holds parameters controlling how reestimation is done.
   **/
  GmmStats(GmmSet& gmmSet, const map<string, string>& params = ParamsType());
```

util.H:
GmmSet实现了GMM模型，并提供了有关gmm算法的实现

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Class holding set of diagonal covariance GMM's.
 *
 *   Here, we summarize the key routines for accessing the parameters of
 *   each GMM.  Each GMM has a set of component Gaussians.  To find the
 *   number of components of a GMM, use #get_component_count().  To find the
 *   mixture weight of a particular component of a GMM,
 *   use #get_component_weight().
 *   To get the means and variances of a particular component, first call
 *   #get_gaussian_index() to find the index of the corresponding Gaussian.
 *   Then, one can call #get_gaussian_mean() and #get_gaussian_var()
 *   with this index to find the means and variances.
 *   The reason for this indirection is to support the sharing of
 *   Gaussians between GMM's.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class GmmSet {
 public:
  /** Ctor; loads from file @p fileName if argument present. **/
  GmmSet(const string& fileName = string());
```

p2.GMM-EM估计

gmm_util.H:
GmmCount用来统计后验概率，用来update gmm参数

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   GMM count class.
 *
 *   Holds a posterior count for a GMM at a frame.  Includes the
 *   GMM index, the frame, and the posterior count.  This is
 *   used to facilitate Forward-Backward and Viterbi EM training.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class GmmCount {
 public:
  /** Ctor; initializes fields to default values. **/
  GmmCount() : m_gmmIdx(0), m_frmIdx(0), m_count(0.0) {}

  /** Ctor; explicitly initializes all fields. **/
  GmmCount(unsigned gmmIdx, unsigned frmIdx, double count)
      : m_gmmIdx(gmmIdx), m_frmIdx(frmIdx), m_count(count) {}
```

p2.GMM-EM估计

lab2_train.C

```
void main_loop(const char** argv) {
  map<string, string> params;
  process_cmd_line(argv, params);

  Lab2TrainMain mainObj(params);
  GmmStats gmmStats(mainObj.get_gmm_set(), params);
  while (mainObj.init_iter()) {
    gmmStats.clear();
    while (mainObj.init_utt()) {
      double logProb =
        gmmStats.update(mainObj.get_gmm_counts(), mainObj.get_feats());
      mainObj.finish_utt(logProb);
    }
    mainObj.finish_iter();
    gmmStats.reestimate();
  }
  mainObj.finish();
}
```

1.根据音频数据和对齐数据进行统计

2.更新GMM参数

p2.GMM-EM估计

gmm_util.C

```
double GmmStats::update(const vector<GmmCount>& gmmCountList,
                        const matrix<double>& feats) {
  unsigned frmCnt = feats.size1();
  unsigned gmmCnt = m_gmmSet.get_gmm_count();
  unsigned lastFrmIdx = (unsigned)-1;
  vector<double> frameBuf;
  double logProb = 0.0;

  for (unsigned cntIdx = 0; cntIdx < gmmCountList.size(); ++cntIdx) {
    const GmmCount& gmmCount = gmmCountList[cntIdx];
    unsigned curFrmIdx = gmmCount.get_frame_index();
    unsigned gmmIdx = gmmCount.get_gmm_index();
    if ((curFrmIdx >= frmCnt) || (gmmIdx >= gmmCnt))
      throw runtime_error(
          "Out-of-bounds frame index or GMM index "
          "in GMM count.");
    if (curFrmIdx != lastFrmIdx)
      copy_matrix_row_to_vector(feats, curFrmIdx, frameBuf);
    logProb += add_gmm_count(gmmIdx, gmmCount.get_count(), frameBuf);
    lastFrmIdx = curFrmIdx;
  }
  return logProb;
}
```

p2.GMM-EM估计

gmm_util.C

GMM计数，需要完成代码的部分

```
double GmmStats::add_gmm_count(unsigned gmmIdx, double posterior,
                               const vector<double>& feats) {
  if (m_gmmSet.get_component_count(gmmIdx) != 1)
    throw runtime_error("GMM doesn't have single component.");
  int gaussIdx = m_gmmSet.get_gaussian_index(gmmIdx, 0);
  int dimCnt = m_gmmSet.get_dim_count();

  //  BEGIN_LAB
  //
  //  Input:
  //      "dimCnt" holds the dimension of the Gaussian and the
  //      acoustic feature vector.
  //      The acoustic feature vector is held in
  //      "feats[0 .. (dimCnt-1)]".
  //      "gaussIdx" is the index of the Gaussian to be updated.
  //      "posterior" is the posterior count of this Gaussian for
  //      the current frame.
  //
  //      The values of the current means and variances can be
  //      accessed via the object "m_gmmSet".
  //
  //  Output:
  //      You should update the counts stored in
```

这里我们根据feats和posterior，将统计值存放在：

```
/** First-order stats for each dim of each Gaussian. **/
matrix<double> m_gaussStats1;

/** Second-order stats for each dim of each Gaussian. **/
matrix<double> m_gaussStats2;
```

gmm_util.H:
class GmmStats

p2.GMM-EM估计

gmm_util.C　　更新GMM参数

```
void GmmStats::reestimate() const {
  //  Reestimate Gaussian means and variances.
  int gaussCnt = m_gmmSet.get_gaussian_count();
  int dimCnt = m_gmmSet.get_dim_count();

  //  BEGIN_LAB
  //
  //  Input:
  //      "gaussCnt" holds the total number of Gaussians.
  //      "dimCnt" holds the dimension of the Gaussians.
  //
  //      The counts you have collected above are stored in:
  //
  //      m_gaussCounts[0 .. (#gaussians-1)]
  //      m_gaussStats1(0 .. (#gaussians-1), 0 .. (dimCnt - 1))
  //      m_gaussStats2(0 .. (#gaussians-1), 0 .. (dimCnt - 1))
  //
  //  Output:
  //      You should call the functions:
  //
  //      m_gmmSet.set_gaussian_mean(gaussIdx, dimIdx, newMean);
  //      m_gmmSet.set_gaussian_var(gaussIdx, dimIdx, newVar);
  //
  //      for each dimension of each Gaussian with the reestimated
  //      values of the means and variances.

  //  END_LAB
  //
```

```
/** First-order stats for each dim of each Gaussian. **/
matrix<double> m_gaussStats1;

/** Second-order stats for each dim of each Gaussian. **/
matrix<double> m_gaussStats2;
```

使用m_gaussStats1和m_gaussStats2中的值，更新属性m_gmmSet

```
/** Reference to associated GmmSet. **/
GmmSet& m_gmmSet;
```

有关GmmSet类的详情可查看util.H util.C

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Class holding set of diagonal covariance GMM's.
 *
 *   Here, we summarize the key routines for accessing the parameters of
 *   each GMM.  Each GMM has a set of component Gaussians.  To find the
 *   number of components of a GMM, use #get_component_count().  To find the
 *   mixture weight of a particular component of a GMM,
 *   use #get_component_weight().
 *   To get the means and variances of a particular component, first call
 *   #get_gaussian_index() to find the index of the corresponding Gaussian.
 *   Then, one can call #get_gaussian_mean() and #get_gaussian_var()
 *   with this index to find the means and variances.
 *   The reason for this indirection is to support the sharing of
 *   Gaussians between GMM's.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class GmmSet {
```

p3

- 用前向后向算法来估计参数，完成src/lab2_fb.C中的两处代码。
- 运行：
  - ./lab2_p3a.sh: 1条数据，1轮迭代
  - ./lab2_p3b.sh: 22条数据，1轮迭代
  - ./lab2_p3c.sh: 22条数据，20轮迭代
  - ./lab2_p3d.sh: 使用p3c的训练的模型，使用viterbi算法解码，结果应该和p1b的结果一样一样
- 比较结果：如p1，分别比较p3a_chart.dat/p3a_chart.ref和p3b.gmm/p3b.gmm.ref。

```bash
#!/bin/bash -e


if [[ -e ./src/lab2_fb ]] ; then
    binStr="./src/lab2_fb"
else
    echo "Couldn't find program to execute."
    exit 1
fi

$binStr --audio_file p018k7.1.dat --graph_file p018k7.1.fsm  --iters 1 \
    --in_gmm p018k7.22.2.gmm --out_gmm /dev/null --chart_file p3a_chart.dat
```

p3.前向后向算法估计GMM

lab2_fb.C

```
void main_loop(const char** argv) {
  map<string, string> params;
  process_cmd_line(argv, params);

  Lab2FbMain mainObj(params);
  GmmStats gmmStats(mainObj.get_gmm_set(), params);
  while (mainObj.init_iter()) {
    gmmStats.clear();
    while (mainObj.init_utt()) {
      double logProb = forward_backward(
          mainObj.get_graph(), mainObj.get_gmm_probs(), mainObj.get_chart(),
          mainObj.get_gmm_counts(), mainObj.get_trans_counts());
      mainObj.finish_utt(logProb);
      gmmStats.update(mainObj.get_gmm_counts(), mainObj.get_feats());
    }
    mainObj.finish_iter();
    gmmStats.reestimate();
  }
  mainObj.finish();
}
```

我们需要完成作业的地方

我们之前p2时完成的函数

p3.前向后向算法估计GMM

lab2_fb.C　　　　作业需要完成代码的部分

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   Routine for the Forward-Backward algorithm.
 *
 *   @param graph HMM/graph to operate on.
 *   @param gmmProbs Matrix of log prob for each GMM for each frame.
 *   @param chart Dynamic programming chart to fill in; already
 *       allocated to be of correct size and initialized with default values.
 *   @param gmmCountList List of GMM counts to be filled in; this vector
 *       will be empty on entry.
 *   @param transCounts Transition/arc counts to be filled in.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
double forward_backward(const Graph& graph, const matrix<double>& gmmProbs,
                        matrix<FbCell>& chart, vector<GmmCount>& gmmCountList,
                        map<int, double>& transCounts) {
  int frmCnt = chart.size1() - 1;
  int stateCnt = chart.size2();
```

p3.前向后向算法估计GMM

lab2_fb.C

作业需要我们完成前向概率，后向概率的计算，并使用两者计算后验概率，这个后验概率会被用来更新GMM参数。

1.我们计算的前向后向概率保存在一个chart中。

```
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
double forward_backward(const Graph& graph, const matrix<double>& gmmProbs,
                        matrix<FbCell>& chart, vector<GmmCount>& gmmCountList,
                        map<int, double>& transCounts) {
  int frmCnt = chart.size1() - 1;
  int stateCnt = chart.size2();
```

FbCell可在.H文件中查看

```
class FbCell {
 public:
  /** Ctor; inits F+B log probs to g_zeroLogProb. **/
  FbCell() : m_forwLogProb(g_zeroLogProb), m_backLogProb(g_zeroLogProb) {}

#ifndef SWIG
#ifndef DOXYGEN
  //  Hack; for bug in matrix<> class in boost 1.32.
  explicit FbCell(int)
      : m_forwLogProb(g_zeroLogProb), m_backLogProb(g_zeroLogProb) {}
#endif
#endif

  /** Sets forward log prob of cell. **/
  void set_forw_log_prob(double logProb) { m_forwLogProb = logProb; }

  /** Sets backward log prob of cell. **/
  void set_back_log_prob(double logProb) { m_backLogProb = logProb; }

  /** Returns forward log prob of cell. **/
  double get_forw_log_prob() const { return m_forwLogProb; }

  /** Returns backward log prob of cell. **/
  double get_back_log_prob() const { return m_backLogProb; }

 private:
  /** Forward logprob. **/
  double m_forwLogProb;

  /** Backward logprob. **/
  double m_backLogProb;
};
```

p3.前向后向算法估计GMM

util.H

在计算前后向概率的时候，
可能会用到util.h中的函数。

```
/** Adds the log probs held in @p logProbList, returning answer as log prob.
 *   That is, let's say we have a list of probability values, the logs of
 *   which are stored in @p logProbList.  Then, this routine returns the
 *   log of the sum of those probability values.
 *   Logarithms are base <i>e</i>.
 **/
double add_log_probs(const vector<double>& logProbList);
```

p3.前向后向算法估计GMM

lab2_fb.C

2.计算完前后向概率后，我们要求后验概率，存放在gmmCountList中。

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
double forward_backward(const Graph& graph, const matrix<double>& gmmProbs,
                        matrix<FbCell>& chart, vector<GmmCount>& gmmCountList,
                        map<int, double>& transCounts) {
  int frmCnt = chart.size1() - 1;
  int stateCnt = chart.size2();
```

后验概率用类GmmCount保存，其在util.H中定义

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 *   GMM count class.
 *
 *   Holds a posterior count for a GMM at a frame.  Includes the
 *   GMM index, the frame, and the posterior count.  This is
 *   used to facilitate Forward-Backward and Viterbi EM training.
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
class GmmCount {
 public:
  /** Ctor; initializes fields to default values. **/
  GmmCount() : m_gmmIdx(0), m_frmIdx(0), m_count(0.0) {}

  /** Ctor; explicitly initializes all fields. **/
  GmmCount(unsigned gmmIdx, unsigned frmIdx, double count)
      : m_gmmIdx(gmmIdx), m_frmIdx(frmIdx), m_count(count) {}

  /** Sets all values in object. **/
  void assign(unsigned gmmIdx, unsigned frmIdx, double count) {
    m_gmmIdx = gmmIdx;
    m_frmIdx = frmIdx;
    m_count = count;
  }

  /** Returns the associated GMM index. **/
  unsigned get_gmm_index() const { return m_gmmIdx; }

  /** Returns the associated frame index. **/
  unsigned get_frame_index() const { return m_frmIdx; }

  /** Returns the posterior count. **/
  double get_count() const { return m_count; }

 private:
  /** The index of the GMM. **/
  unsigned m_gmmIdx;

  /** Which frame the count occurred at. **/
  unsigned m_frmIdx;

  /** The posterior count. **/
  float m_count;
};
```

p3.前向后向算法估计GMM

当我们有了对应的gmmId，
数据桢，后验概率m_count，
我们就可以利用p2中我们
完成的算法去更新gmm的
值。

```
private:
 /** The index of the GMM. **/
 unsigned m_gmmIdx;

 /** Which frame the count occurred at. **/
 unsigned m_frmIdx;

 /** The posterior count. **/
 float m_count;
```

谢谢