Kaldi训练三音素模型

> 主要是kaldi 中的aishell 中的例子。注意aishell 中的 `steps/` 和 `utils/` 主要是wsj例子中的，这里不做区分

1. 三音素模型原理

首先介绍一下三音素模型。

1.1 三音素模型

1. 为什么要用三音素模型?

   单音素建模没有考虑协同发音效应，也就是上下文音素会对当前的中心音素发音有影响，会产生协同变化，这与该音素的单独发音会有所不同(数据统计也就有所不同)，为了考虑这个影响，所以需要使用三音素建模，使得模型描述更加精准。

2. 三音素的问题
   - 模型中参数众多，效率太低
   - 参数众多导致数据量要足够大，否则无法学习到正确的参数

那么如何使用三音素但是可以减少参数量呢?

- smoothing ： 讲三音素进行smoothing，使得left和right值的影响合到mid的值上（个人的理解啊，不知道对不对，因为也没有找到对应的代码作证）
- clustering / parameter-sharing

  当前中心音素，如果上下文的发音类型相似，则对当前音素的影响是相似的，则可以将这些数据聚为一类。

所以问题的关键就变为了，如何定义*相似*? 如何划分出*相似*?

1.2 决策树

核心思想：构建树以方便将相似的三音素状态聚类为一个状态以减少参数

- 根节点存储所有状态集合
- 通过不断的划分，最终叶子结点为最终分类的簇
- 在每个分支中进行数据划分，直到达到不划分的准则

> 停止划分一般是:
>
> 1. 评价指标无法通过划分变得足够好
> 2. 数据划分到此太少了，再划分将不再具有代表性

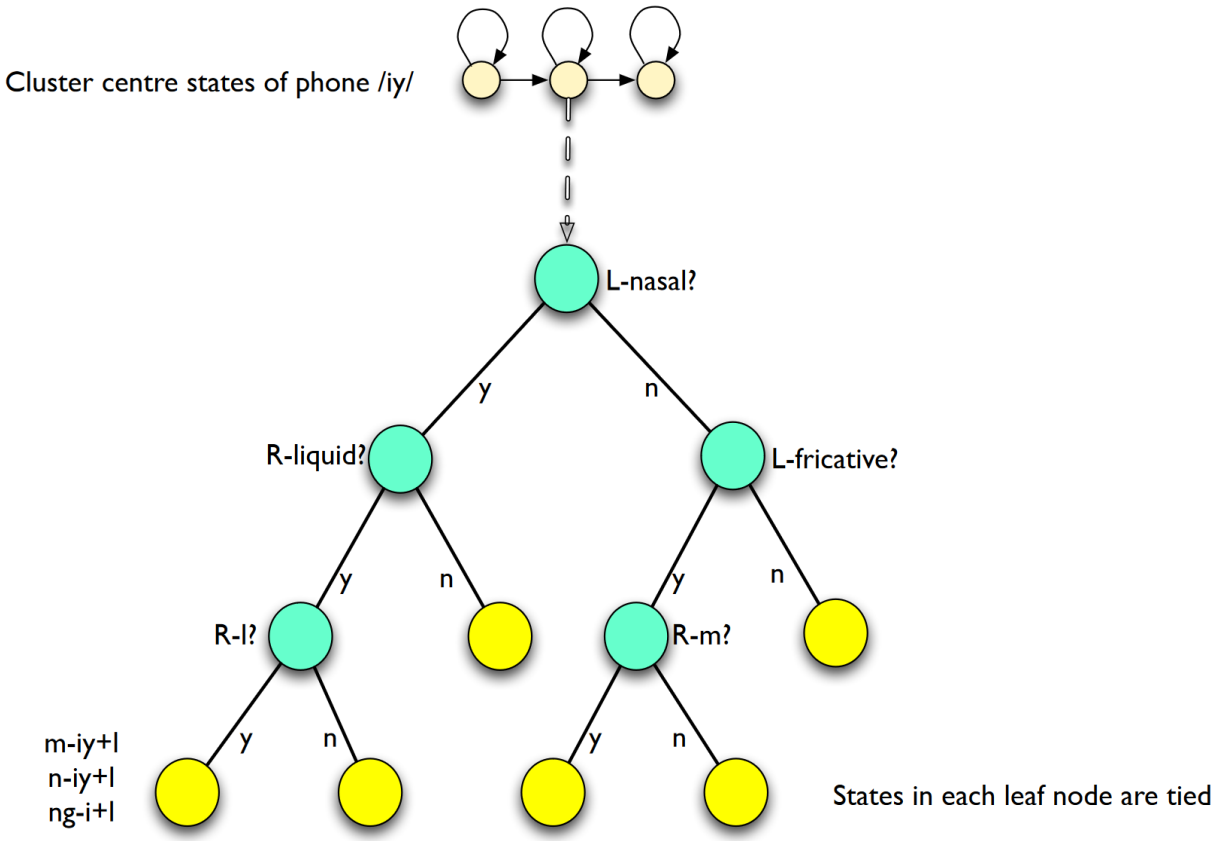
关键在于：划分准则

当到达一个结点的时候，如何划分这个结点。

具体要如何制定这些规则(决策树规则)，靠语言发音学家的经验知识，提问。(音素判别，再到状态绑定)对于节点分裂，需要寻找最佳的问题，按照looklikehood增加的原则:

$$\Delta = L(S_y) + L(S_n) - L(s)$$

以下是一些经典的问题:



> kaldi可以自动产生问题集，根据音素本身数据上的相似性，自动聚为一类，这不需要语言学知识，而是直接利用数据学习划分的标准（其实Kaldi这种方法更加贴合决策树本来的做法，原本的决策树就是贪婪的利用信息熵或者基尼指数来使得数据划分的纯度变得最好）。我们将在后续建树的过程中看到这一点。

2. kaldi 实现

在kaldi 中，三音素模型这里的shell 脚本主要是 `train_deltas.sh` ，这个脚本的调用如下:

```
steps/train_deltas.sh --cmd "$train_cmd" \
 2500 20000 data/train data/lang exp/tri1_ali exp/tri2 || exit 1;
```

参数的意义在脚本中可以看到如下:

```
numleaves=$1 # 2500
totgauss=$2  # 20000
data=$3      # data/train
lang=$4      # data/lang
alidir=$5    # exp/tri1_ali
dir=$6       # exp/tri2
```

需要的数据是：

```
$alidir/final.mdl $alidir/ali.1.gz $data/feats.scp $lang/phones.txt
```

2.0 Decision tree internals 综述

2.0.1 Event maps

这里的Event并不是指事件，其表示的是key-value对，注意最终使用结构来存储的时候key是不重复的。这里想到的是利用C++ STL 的std::Map<int32,int32>

> 注意int32 是 `kaldi::int32`

但是Map 的效率往往不是那么好，所以在Kaldi中定义为一个vector，如下：

```
/* event-map.h */
/// Things of type EventKeyType can take any value.  The code does not assume they are contiguous.
/// So values like -1, 1000000 and the like are acceptable.
typedef int32 EventKeyType;

/// Given current code, things of type EventValueType should generally be nonnegative and in a
/// reasonably small range (e.g. not one million), as we sometimes construct vectors of the size:
/// [largest value we saw for this key].  This deficiency may be fixed in future [would require
/// modifying TableEventMap]
typedef int32 EventValueType;

/// As far as the event-map code itself is concerned, things of type EventAnswerType may take
/// any value except kNoAnswer (== -1).  However, some specific uses of EventMap (e.g. in
/// build-tree-utils.h) assume these quantities are nonnegative.
typedef int32 EventAnswerType;

typedef std::vector<std::pair<EventKeyType, EventValueType> > EventType;
```

这里的EventKeyType,EventValueType都被定义为int32，同样使用一个typedef即可，后面的EventAnswerType也是同样的道理。表示一个pdf标识符（声学状态索引）。

EventMap的解释这里我们看一个官方教程中的例子：

> 我们的上下文音素是 a/b/c ,假设我们有一个标准的三音素拓扑结构；和假设在这个上下文中我们想问音素"b"的中心状态的pdf对应的索引是多少。这个问题里这个状态将是状态1，因为我们使用从0开始的索引。当我们说到"state"，我们将跳过一些细节；更多的细节你可以看Pdf-classes。假设音素a，b和c的整数 索引分别为10，11和12。"event"对应的映射为：

> `(0->10),(1->11),(2->12),(-1->1)`
>
> 在三音素窗"a/b/c"的0，1和2是位置，和-1是一个特殊的索引，我们用这个来编码state-id(请 参照：常量kPdfClass = -1)的一个特殊的索引。用这种方式的话，我们的一个排好序的向量对 是：
>
> `EventType e = {{-1,1},{0,10},{1,11},{2,12}};`
>
> 因此我们认为EventMap是从EventType到EventAnswerType的一个映射，你可以认为这是一个从上下文音素到整数索引的一个映射。

注意EventMap 是一个abstract的类，无法直接初始化，因此kaldi中给出三个具体的类，在实际的使用中，我们用多态的特性这样非常方便优雅的控制了结点的类型：

1. ConstantEventMap: 认为它是一个决策树叶子结点。这个类存储类型EventAnswerType 的一个整数和它的映射函数常常返回这个值。
2. SplitEventMap:认为它是一个决策树的非叶子结点，它查询一个特定的key，和根据问题 的答案去"yes"或者"no"孩子结点。
3. TableEventMap:这个在一个特定的key上做一个完整的分裂。一个典型的例子就是：你 也许首先完全分裂中心音素，然后你对这个音素的每一值有一个分离的决策树。内部地 它存储EventMap*指针的一个向量。它查找对应分裂的key的值，和调用向量对应位置的 EventMap的映射函数。

以上这些定义基本上都在event-map.h这个文件当中。

我们将在建树的过程中看到这些定义的作用。

2.0.2 Statistics for building the tree

用来建立音素决策树的统计量如下：

```
1  typedef std::vector<std::pair<EventType,Clusterable*> > BuildTreeStatsType;
```

这种类型的一个对象的引用被传递到所有决策树建立过程中。这些统计量表示一个从EventType到Clusterable的映射。在我们目前的代码中Clusterable对象是一个真正的类型GaussClusterable。注意树建立的过程无所谓是GaussClusterable还是其他的，只需要实现了Clusterable即可。积累这些统计量的程序是acc-tree-stats.cc。

2.1 `acc-tree-stats.cc`

> Accumulate tree statistics for decision tree training 为了构建决策树累积相关的统计量。

在文件中，可以看到一个调用(注意在实际的调用中，该命令是在 `JOB=1:nj` 的情况下的，所以不是只调用了一次，生成的文件是 `JOB.treeacc` 格式的，这一点将在后面的sum过程中使用到)：

`acc-tree-stats 1.mdl scp:train.scp ark:1.ali 1.tacc`

可以看到这几个参数是什么意思:

```
1  model_filename = po.GetArg(1); # 声学模型
2  feature_rspecifier = po.GetArg(2); # 特征
3  alignment_rspecifier = po.GetArg(3); # 对齐
4  accs_out_wxfilename = po.GetOptArg(4); # 最后输出的文件名
```

首先看一下前面没有涉及到的类，基本上都是为了读取特定格式的文件：

1. ParseOptions

   用于转化命令行的函数，这是Kaldi 自带的，不属于C++，构造函数传入usage字符串以说明用法。

2. SequentialBaseFloatMatrixReader

   为了读取特征矩阵。

3. RandomAccessInt32VectorReader

   为了读取对齐信息。

首先看一下核心过程（不是完整代码，删除了中间一些判断有效和断言的语句（这里的断言写到log中的方法非常精妙，大家可以在以后的C++代码中借鉴），下面的代码都做了类似的处理）：

```
// 加载参数
const char *usage =
        "Accumulate statistics for phonetic-context tree building.\n"
        "Usage:  acc-tree-stats [options] <model-in> <features-rspecifier> <alignments-rspecifier> <tree-accs-out>\n"
        "e.g.: \n"
        " acc-tree-stats 1.mdl scp:train.scp ark:1.ali 1.tacc\n";
bool binary = true;
AccumulateTreeStatsOptions opts;
ParseOptions po(usage);
po.Register("binary", &binary, "Write output in binary mode");
opts.Register(&po);
po.Read(argc, argv);
std::string model_filename = po.GetArg(1),
        feature_rspecifier = po.GetArg(2),
        alignment_rspecifier = po.GetArg(3),
        accs_out_wxfilename = po.GetOptArg(4);
AccumulateTreeStatsInfo acc_tree_stats_info(opts);
TransitionModel trans_model;
SequentialBaseFloatMatrixReader feature_reader(feature_rspecifier);
RandomAccessInt32VectorReader alignment_reader(alignment_rspecifier);

// 定义tree_stats并利用特征合对齐生成BuildTreeStatsType
std::map<EventType, GaussClusterable*> tree_stats;
// 统计完成和失败的个数
int num_done = 0, num_no_alignment = 0, num_other_error = 0;

// 这个地方类似于一种while(next())的写法
for (; !feature_reader.Done(); feature_reader.Next()) {
    std::string key = feature_reader.Key();
    // 对齐数据没有找到对应的特征，注意本程序最后
    if (!alignment_reader.HasKey(key)) {
        num_no_alignment++;
    } else {
        const Matrix<BaseFloat> &mat = feature_reader.Value();
        const std::vector<int32> &alignment = alignment_reader.Value(key);
        if (alignment.size() != mat.NumRows()) {
            KALDI_WARN << "Alignments has wrong size "<< (alignment.size())<<" vs. "<< (mat.NumRows());
            num_other_error++;
            continue;
        }

```

```
        AccumulateTreeStats(trans_model,
                            acc_tree_stats_info,
                            alignment,
                            mat,
                            &tree_stats);
        num_done++;
        if (num_done % 1000 == 0) // 每处理完1000条数据打印一个Log
            KALDI_LOG << "Processed " << num_done << " utterances.";
    }
}

BuildTreeStatsType stats;
// 遍历tree_stats
for (std::map<EventType, GaussClusterable*>::const_iterator iter = tree_stats.begin(); iter
!= tree_stats.end();++iter) {
    stats.push_back(std::make_pair(iter->first, iter->second));
}
// 这里有打印操作，因为比较简单没有留在这里
// 最后清空
tree_stats.clear();
DeleteBuildTreeStats(&stats);
if (num_done != 0) return 0;
else return 1;
```

注意其中的GaussClusterable 是继承自 Clusterable 所以尽管在前面可以看到在前面的BuildTreeStatsType的定义中类型不同，但是放入这个类中这个操作依然是可行的。

然后我们来看一下核心函数 `AccumulateTreeStats()`：

```
void AccumulateTreeStats(const TransitionModel &trans_model,
                         const AccumulateTreeStatsInfo &info,
                         const std::vector<int32> &alignment,
                         const Matrix<BaseFloat> &features,
                         std::map<EventType, GaussClusterable *> *stats)
{
  std::vector<std::vector<int32>> split_alignment;
  // 解释以下SplitTiPhones
  // 这个函数就是利用trans_model和alignment 将一段相同的phones截取出来，并存储到split_alignment
中
  bool ans = SplitToPhones(trans_model, alignment, &split_alignment);

  int32 cur_pos = 0;
  int32 dim = features.NumCols();

  for (int32 i = -info.context_width; i < static_cast<int32>(split_alignment.size()); i++)
  {
    // consider window starting at i, only if i+info.central_position is within
    // list of phones.
    if (i + info.central_position >= 0 &&
        i + info.central_position < static_cast<int32>(split_alignment.size()))
    {
      int32 central_phone =
          MapPhone(info.phone_map,
                   trans_model.TransitionIdToPhone(
                       split_alignment[i + info.central_position][0]));
      bool is_ctx_dep = !std::binary_search(info.ci_phones.begin(),
                                            info.ci_phones.end(),
```

```
                                            central_phone);
      EventType evec;


      for (int32 j = 0; j < info.context_width; j++)
      {
        int32 phone;
        if (i + j >= 0 && i + j < static_cast<int32>(split_alignment.size()))
          phone =
              MapPhone(info.phone_map,
                       trans_model.TransitionIdToPhone(split_alignment[i + j][0]));
        else
          phone = 0;
        if (is_ctx_dep || j == info.central_position)
          evec.push_back(std::make_pair(static_cast<EventKeyType>(j),
static_cast<EventValueType>(phone)));
      }
      //
      for (int32 j = 0; j < static_cast<int32>(split_alignment[i +
info.central_position].size()); j++)
      {
        // for central phone of this window...
        EventType evec_more(evec);
        int32 pdf_class = trans_model.TransitionIdToPdfClass(
            split_alignment[i + info.central_position][j]);
        // pdf_class will normally by 0, 1 or 2 for 3-state HMM.
        std::pair<EventKeyType, EventValueType> pr(kPdfClass, pdf_class);
        evec_more.push_back(pr);
        std::sort(evec_more.begin(), evec_more.end()); // these must be sorted!
        if (stats->count(evec_more) == 0)
          (*stats)[evec_more] = new GaussClusterable(dim, info.var_floor);

        BaseFloat weight = 1.0;
        (*stats)[evec_more]->AddStats(features.Row(cur_pos), weight);
        cur_pos++;
      }
    }
  }
}
```

注：

1. **TransitionModel**

   kaldi中的HMM模型，实际就是一个TransitionModel对象。这个对象描述了音素的HMM拓扑结构，并保存了pdf-id和transition-id相关的信息，并且可以进行各种变量的转换。

   在介绍TransitionModel之前，先介绍一些概念。

   - phone: 音素，从1开始编号。可以根据phones.txt映射为具体音素
   - HMM-state: 音素HMM模型的状态，从0开始编号
   - pdf-id: 决策树和声学模型中用到的pdf的编号，从0开始
   - transition-state: 一个（虚拟的）状态，通过弧跳转到自己或其他状态。某些情况下，可以跟pdf-id一一对应。
   - transition-index: HMM状态中转移的索引，即HmmTopology::HmmState::transitions的索引，从0开始编号
   - transition-id: 所有的HMM状态的弧进行编号。从1开始编号。

   通常，将phone、HMM-state和pdf-id（包括forward-pdf-id, self-loop-pdf-id）作为一个元组（Tuple），一个元组，可映射为一个transition-state。transition-state加一个具体的transition-index，可以映射出一个transition-id。

   简单而言就是下面这样:

> (phone, HMM-state, forward-pdf-id, self-loop-pdf-id) -> transition-state
>
> (transition-state, transition-index) -> transition-id

可以参考这个CSDN博客

2. **AccumulateTreeStatsInfo**

   如果详细看的话可以发现:

```
/* tree-accu.h */
info.context_width == 3;
info.central_position == 1;
info.ci_phones ; // 读取phones 文件sort得到的结果$lang/phones/context_indep.csl
```

   也就是说直接满足三音素的定义

3. **SplitToPhones()和SplitToPhonesInternal()**

   实际运行的是SplitToPhonesInternal()函数，作用是把相同的phones对应的状态存在同一个vector当中，将这些vector存储到split_output中。

4. 循环变量j

   在i循环中的循环变量j两次循环的意义是不同的:

   - 第一次，j变量表示的是遍历整个三音素的下标，注意到变量i总是表示一个三音素的第一个音素的位置，所以这里，i+j就是表示三音素中的某个音素的位置。这样可以解释 `i + central_position` 的意义
   - 第二次，j变量表示的遍历某个音素的全部状态，所以其上限才是 `split_alignment[i + info.central_position].size()`

   通过这两次遍历就把三音素整合到一个EventMap中了，这样方便了存储和后续的操作

2.2 `sum-tree-stats.cc`

> 主要是为了统计所有上下文决策树的生成相关的数据，其中核心操作就是把原来的各个信息整合到一起

在文件中调用的方式如下:

```
sum-tree-stats treeacc 1.treeacc 2.treeacc 3.treeacc ...
```

简单看一下代码（逻辑比较简单，不详细说了，这里的iterator迭代器的写法实现的非常好，感兴趣的同学可以详细看一下）:

```
// 进行
const char *usage =
    "Sum statistics for phonetic-context tree building.\n"
    "Usage:  sum-tree-stats [options] tree-accs-out tree-accs-in1 tree-accs-in2 ...\n"
    "e.g.: \n"
    " sum-tree-stats treeacc 1.treeacc 2.treeacc 3.treeacc\n";

ParseOptions po(usage);
bool binary = true;
po.Register("binary", &binary, "Write output in binary mode");
po.Read(argc, argv);
```

```
12
13  std::map<EventType, Clusterable *> tree_stats;
14
15  std::string tree_stats_wxfilename = po.GetArg(1);
16
17  // A reminder on what BuildTreeStatsType is:
18  // typedef std::vector<std::pair<EventType, Clusterable*> > BuildTreeStatsType;
19  // 注意到上述的参数中，从第二个参数开始是acc-tree-stats 生成的单个的结果
20  for (int32 arg = 2; arg <= po.NumArgs(); arg++)
21  {
22    std::string tree_stats_rxfilename = po.GetArg(arg);
23    bool binary_in;
24    Input ki(tree_stats_rxfilename, &binary_in);
25    BuildTreeStatsType stats_array;
26    GaussClusterable example; // Lets ReadBuildTreeStats know which type to read..
27    ReadBuildTreeStats(ki.Stream(), binary_in, example, &stats_array);
28    for (BuildTreeStatsType::iterator iter = stats_array.begin();
29          iter != stats_array.end(); ++iter)
30    {
31      EventType e = iter->first;
32      Clusterable *c = iter->second;
33      // 查看是否在map中已经出现过了。
34      std::map<EventType, Clusterable *>::iterator map_iter = tree_stats.find(e);
35      if (map_iter == tree_stats.end())
36      {
37        tree_stats[e] = c;
38      }
39      else
40      {
41        map_iter->second->Add(*c);
42        delete c;
43      }
44    }
45  }
46
47  BuildTreeStatsType stats; // vectorized form.
48  // 写入一个总和的文件中
49  // 之所以会这么折腾就是因为这个文件输出函数`WriteBuildTreeStats()`貌似实现的不是非常好
50  // 可以使用auto iterator 的，这个地方应该是考虑了实现方便吧
51  for (std::map<EventType, Clusterable *>::const_iterator iter = tree_stats.begin();
52        iter != tree_stats.end();
53        ++iter)
54  {
55    stats.push_back(std::make_pair(iter->first, iter->second));
56  }
57  // 取消内存占用
58  tree_stats.clear();
59  DeleteBuildTreeStats(&stats);
```

这个函数还是比较简单的。

到目前为止，我们已经成功的讲特征信息、对齐、单音素训练下的数据进行了统一，并且也对于三音素的数据进行了非常方便的处理，下面的工作将基于当前的数据结构和格式。

解释几个疑问：

- 为什么要不断的std::Map和std::vector不断的进行转化

  个人的理解是因为vector对于序列操作更加方便，std::map取最后的值，遍历等等都比较不方便，但是vector可以和方便的对于序列进行扩充。

  当然这样操作的风险也是存在的，可以看到在后面的代码中将对于vector进行不断的排序，这就是因为map原本是有序的数据结构RB-Tree，而不排序的vector（原本应该是map）就会被识别两个，这就需要编程人员足够的小心。

// 下面的先没有做

2.3 `cluster-phones.cc`

> 对于phones 进行聚类，最终得到一个问题集

```
cluster-phones $context_opts $dir/treeacc $lang/phones/sets.int \
    $dir/questions.int 2> $dir/log/questions.log || exit 1;
```

由于上面的 `sum-acc-stats` 最后输出的文件叫treeacc

这个命令获得参数结果如下:

```
std::string stats_rxfilename = po.GetArg(1),
        phone_sets_rxfilename = po.GetArg(2),
        phone_sets_wxfilename = po.GetArg(3);
```

cluster-phone 的过程如下：

- 加载参数
- 从treeacc 文件中读取先前存储好的数据，这里就是BuildTreeStatsType
- 读取pdf_class_list
- 读取音素集合phones_sets
- 根据mode 生成对应的 问题集
- 写入对应的文件

注意本代码中一个关键的参数是 `mode`

```
/* cluster-phone.cc */
if (mode == "questions") {
  if (num_classes != -1)
  AutomaticallyObtainQuestions(stats,
                               phone_sets,
                               pdf_class_list,
                               P,
                               &phone_sets_out);
} else if (mode == "k-means") {
  if (num_classes <= 1 ||
      static_cast<size_t>(num_classes) > phone_sets.size())
  KMeansClusterPhones(stats,
                      phone_sets,
                      pdf_class_list,
                      P,
                      num_classes,
```

```
                      &phone_sets_out);
}
```

以下讨论默认mode=="questions"

关键的函数是 `AutomaticallyObtainQuestions()`

> 它把音素聚类成一棵树，并且对树中的每一个结点，把从该结点可以到达的所有叶子结点合在一起构成一个问题（该树的一个叶子结点保存着一些音素，一个问题就是一个音素的集合）。

```
/* tree/build-tree.cc */
void AutomaticallyObtainQuestions(BuildTreeStatsType &stats,
                                  const std::vector<std::vector<int32>> &phone_sets_in,
                                  const std::vector<int32> &all_pdf_classes_in,
                                  int32 P,
                                  std::vector<std::vector<int32>> *questions_out)
{
  std::vector<std::vector<int32>> phone_sets(phone_sets_in);
  std::vector<int32> phones;

  // 获取phone_sets_in中的结果
  // 对于每个音素对应的观测值进行排序
  // 重新写到phones 中
  for (size_t i = 0; i < phone_sets.size(); i++)
  {
    std::sort(phone_sets[i].begin(), phone_sets[i].end());
    for (size_t j = 0; j < phone_sets[i].size(); j++)
      phones.push_back(phone_sets[i][j]);
  }
  std::sort(phones.begin(), phones.end());

  // 对于pdf_class_in 进行排序并删除重叠的。
  std::vector<int32> all_pdf_classes(all_pdf_classes_in);
  SortAndUniq(&all_pdf_classes);

  BuildTreeStatsType retained_stats;
  // 过滤出在all_pdf_classes 中出现的 value
  // 由于all_pdf_classes默认是1，因此这里取三音素的中间值
  FilterStatsByKey(stats, kPdfClass, all_pdf_classes,
                   true, // retain only the listed positions
                   &retained_stats);


  std::vector<BuildTreeStatsType> split_stats; // split by phone.
  SplitStatsByKey(retained_stats, P, &split_stats);

  std::vector<Clusterable *> summed_stats; // summed up by phone.
  SumStatsVec(split_stats, &summed_stats);

  int32 max_phone = phones.back();
  if (static_cast<int32>(summed_stats.size()) < max_phone + 1)
  {
    // this can happen if the last phone had no data.. if we are using
    // stress-marked, position-marked phones, this can happen.  The later
    // code will assume that a summed_stats entry exists for all phones.
    summed_stats.resize(max_phone + 1, NULL);
  }

  for (int32 i = 0; static_cast<size_t>(i) < summed_stats.size(); i++)
```

```
50    { // A check.
51      if (summed_stats[i] != NULL &&
52          !binary_search(phones.begin(), phones.end(), i))
53      {
54        KALDI_WARN << "Phone " << i << " is present in stats but is not in phone list [make
      sure you intended this].";
55      }
56    }
57
58    EnsureClusterableVectorNotNull(&summed_stats); // make sure no NULL pointers in
      summed_stats.
59    // will replace them with pointers to empty stats.
60
61    std::vector<Clusterable *> summed_stats_per_set(phone_sets.size(), NULL); // summed up
      by set.
62    for (size_t i = 0; i < phone_sets.size(); i++)
63    {
64      const std::vector<int32> &this_set = phone_sets[i];
65      summed_stats_per_set[i] = summed_stats[this_set[0]]->Copy();
66      for (size_t j = 1; j < this_set.size(); j++)
67        summed_stats_per_set[i]->Add(*(summed_stats[this_set[j]]));
68    }
69
70    int32 num_no_data = 0;
71    for (size_t i = 0; i < summed_stats_per_set.size(); i++)
72    { // A check.
73      if (summed_stats_per_set[i]->Normalizer() == 0.0)
74      {
75        num_no_data++;
76        std::ostringstream ss;
77        ss << "AutomaticallyObtainQuestions: no stats available for phone set: ";
78        for (size_t j = 0; j < phone_sets[i].size(); j++)
79          ss << phone_sets[i][j] << ' ';
80        KALDI_WARN << ss.str();
81      }
82    }
83    if (num_no_data + 1 >= summed_stats_per_set.size())
84    {
85      std::ostringstream ss;
86      for (size_t i = 0; i < all_pdf_classes.size(); i++)
87        ss << all_pdf_classes[i] << ' ';
88      KALDI_WARN << "All or all but one of your classes of phones had no data. "
89                 << "Note that we only consider data where pdf-class is in the "
90                 << "set ( " << ss.str() << ").  If you have an unusual HMM "
91                 << "topology this may not be what you want; use the "
92                 << "--pdf-class-list option to change this if needed. See "
93                 << "also any warnings above.";
94    }
95
96    TreeClusterOptions topts;
97    topts.kmeans_cfg.num_tries = 10; // This is a slow-but-accurate setting,
98    // we do it this way since there are typically few phones.
99
100   std::vector<int32> assignments;       // assignment of phones to clusters. dim ==
      summed_stats.size().
101   std::vector<int32> clust_assignments; // Parent of each cluster.  Dim == #clusters.
102   int32 num_leaves;                     // number of leaf-level clusters.
103   TreeCluster(summed_stats_per_set,
104               summed_stats_per_set.size(), // max-#clust is all of the points.
105               NULL,                        // don't need the clusters out.
```

```
106                 &assignments,
107                 &clust_assignments,
108                 &num_leaves,
109                 topts);
110
111     // process the information obtained by TreeCluster into the
112     // form we want at output.
113     ObtainSetsOfPhones(phone_sets,
114                        assignments,
115                        clust_assignments,
116                        num_leaves,
117                        questions_out);
118
119     // The memory in summed_stats was newly allocated. [the other algorithms
120     // used here do not allocate].
121     DeletePointers(&summed_stats);
122     DeletePointers(&summed_stats_per_set);
123   }
124
```

2.4 `compile-questions.cc`

> 分析决策树决策过程的划分条件。

2.5 `build-tree.cc`