Speech Recognition

- a practical guide

Lecture 3

Phonetic Context Dependency

Steps covered in this lecture

```
$ cd ~/kaldi-trunk/rm/s3/
$
$ # Get alignments from monophone system.
$ steps/align_deltas.sh data/train data/lang exp/mono exp/mono_ali
$
$ # train tri1 [first triphone pass]
$ steps/train_deltas.sh data/train data/lang exp/mono_ali exp/tri1
$
```

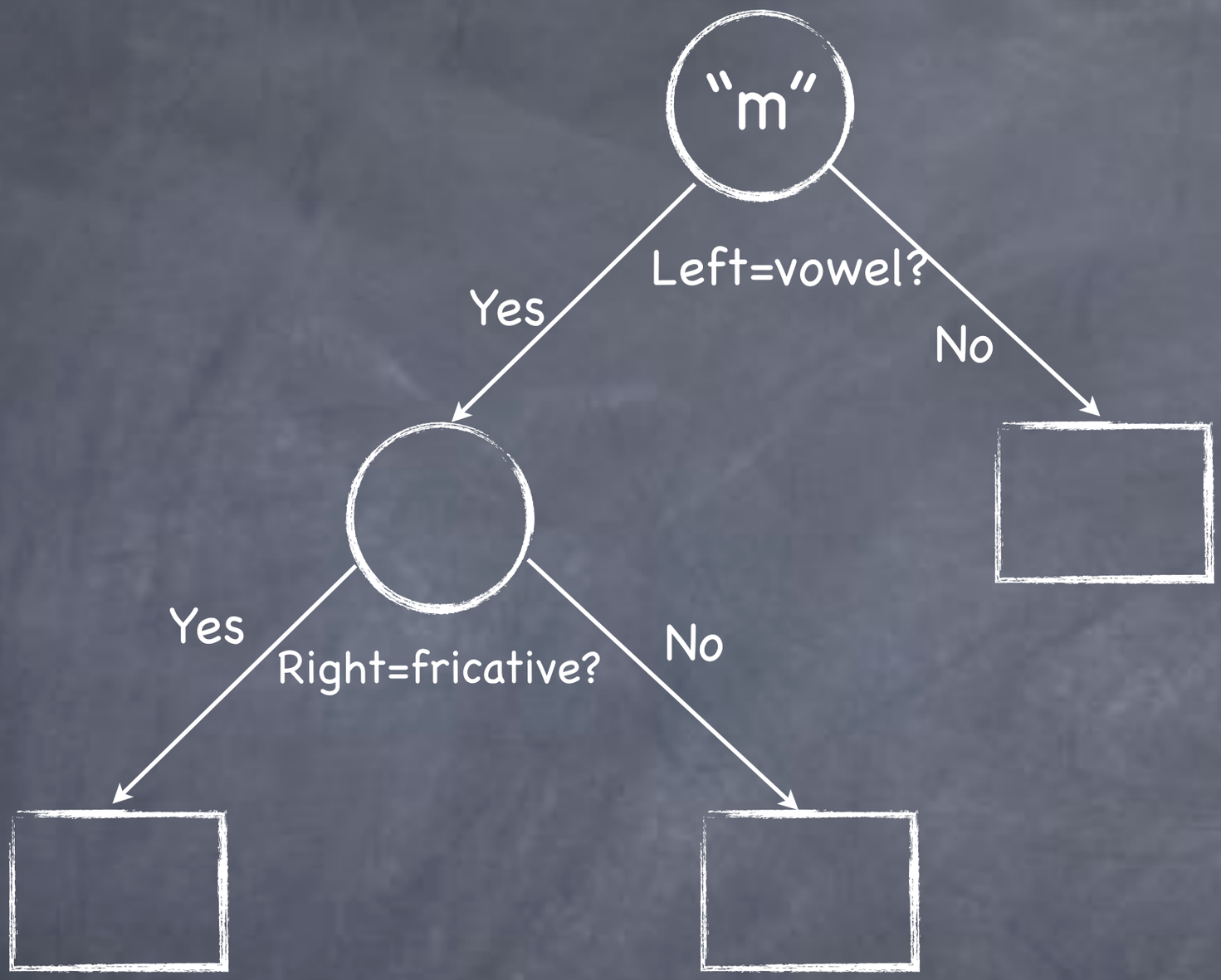
- Aligning data with monophone system
- Training triphone system

Weakness of "monophone" model

- Phones "sound different" in different contexts.
- Most strongly affected by phones immediately before/after.
- Simplest model of context dependency is to build separate model per "triphone" context.
- For 38 phones, #models required is 38x38x38
- Too many models to train!

Traditional context-dependency tree

- Build a "decision tree" for each "monophone"
- This follows the "Clustering and Regression Tree" (CART) framework
  - Involves a "greedy" (locally optimal) splitting algorithm.
- Ask questions like "Is the left phone a vowel?" Is the right phone the phone "sh"?
- Models (HMMs) would correspond to the leaves



Square boxes correspond to Hidden Markov Models

Traditional tree-building

- Train a monophone system (or use previously built triphone system) to get time alignments for data.
- For each seen triphone, accumulate sufficient statistics to train a single Gaussian per HMM state
- Suff. stats for Gaussian are (count, sum, sum-squared).
- Total stats size (for 39-dim feats):
  - (38x38x38) * (3 HMM-states) * (39+39+1)

But not all triphones seen!

Building the triphone system

```
$ cd ~/kaldi-trunk/rm/s3/
$
$ # Get alignments from monophone system.
$ steps/align_deltas.sh data/train data/lang exp/mono exp/mono_ali
$
$ # train tri1 [first triphone pass]
$ steps/train_deltas.sh data/train data/lang exp/mono_ali exp/tri1
$
```

- Align more of the training data using the monophone model.
- We saw the alignment format last time (sequences of integers)
- Note: exp/mono has alignments too, but need it on more data (and with very last model).

Getting data alignments

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ head -1 exp/mono_ali/align.log
gmm-align --transition-scale=1.0 --acoustic-scale=0.1 --self-loop-
scale=0.1 --beam=8 --retry-beam=40 exp/mono_ali/tree exp/mono_ali/
final.mdl data/lang/L.fst 'ark:apply-cmvn --norm-vars=false --
utt2spk=ark:data/train/utt2spk ark:exp/mono_ali/cmvn.ark scp:data/train/
feats.scp ark:- | add-deltas ark:- ark:- |' ark:exp/mono_ali/train.tra
ark:exp/mono_ali/ali
```

- Alignments go in file "exp/mono_ali/ali"
- --beam=8, --retry-beam=40 important options
- For Viterbi pruning
  - If don't reach end-state with beam=8, retry with beam=40, then give up.

Building the triphone system

```
$ cd ~/kaldi-trunk/rm/s3/
$
$ # Get alignments from monophone system.
$ steps/align_deltas.sh data/train data/lang exp/mono exp/mono_ali
$
$ # train tri1 [first triphone pass]
$ steps/train_deltas.sh data/train data/lang exp/mono_ali exp/tri1
$
```

- Rest of lecture will be about this stage.

Getting stats for tree

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ ls -l exp/tri1/treeacc
-rw-r--r-- 1 dpovey clsp 14M Feb  7 20:57 exp/tri1/treeacc
$ cat exp/tri1/acc.tree.log
acc-tree-stats --ci-phones=48 exp/mono_ali/final.mdl 'ark:apply-cmvn --norm-vars=false --
utt2spk=ark:data/train/utt2spk ark:exp/mono_ali/cmvn.ark scp:data/train/feats.scp ark:- |
add-deltas ark:- ark:- |' ark:exp/mono_ali/ali exp/tri1/treeacc
apply-cmvn --norm-vars=false --utt2spk=ark:data/train/utt2spk ark:exp/mono_ali/cmvn.ark
scp:data/train/feats.scp ark:-
add-deltas ark:- ark:-
LOG (acc-tree-stats:main():acc-tree-stats.cc:113) Processed 1000 utterances.
LOG (acc-tree-stats:main():acc-tree-stats.cc:113) Processed 2000 utterances.
LOG (acc-tree-stats:main():acc-tree-stats.cc:113) Processed 3000 utterances.
```

- Double-precision stats.
- Max. possible size is 38^3*3*(39+39+1) * 8 = 104M
- Actual size 14M (~15% of triphones seen).

Looking at tree stats

```
$ . path.sh
$ sum-tree-stats --binary=false - exp/tri1/treeacc | less
BTS 19268
EV 4 -1 0 0 0 1 10 2 22
T GCL 10 0.01  [
  -319.4122 -47.78007 221.1587 158.8115 335.7153 192.0782 20.63128 99.35792 -48.65273
71.48264 -30.99772 -121.76 16.75381 9.745302 26.61232 20.03452 31.32775 25.1797 8.662501
-11.21577 -10.93547 -5.706181 -8.542741 -17.79073 -14.53732 -4.358661 3.026504 -3.41487
-4.261815 -2.914604 -8.994694 -2.071367 -0.131388 3.215806 2.529252 1.733106 1.000433
2.846173 5.605232
  10381.26 1351.984 5520.843 4398.474 12389.38 4130.489 488.7585 1483.611 477.1437 822.2339
1032.555 1925.231 277.2069 19.61546 131.6582 93.57193 264.0651 142.572 61.74911 31.82133
38.00629 46.85469 62.60578 108.7827 51.53504 58.33066 7.887114 18.35285 18.01356 37.78245
```

- This data is for HMM-state 0, phonetic context (0,10,22), = "<eps>/d/ih" (as in "DID" at start of sentence).
- Count of 1st state is 10
- Next two lines are data sum, sum-squared.

Looking at tree stats

```
$ . path.sh
$ sum-tree-stats --binary=false - exp/tri1/treeacc | less
BTS 19268
EV 4 -1 0 0 0 1 10 2 22
T GCL 10 0.01  [
  -319.4122 -47.78007 221.1587 158.8115 335.7153 192.0782 20.63128 99.35792 -48.65273
71.48264 -30.99772 -121.76 16.75381 9.745302 26.61232 20.03452 31.32775 25.1797 8.662501
-11.21577 -10.93547 -5.706181 -8.542741 -17.79073 -14.53732 -4.358661 3.026504 -3.41487
-4.261815 -2.914604 -8.994694 -2.071367 -0.131388 3.215806 2.529252 1.733106 1.000433
2.846173 5.605232
  10381.26 1351.984 5520.843 4398.474 12389.38 4130.489 488.7585 1483.611 477.1437 822.2339
```

- There are 19268 states with stats (this is 3x #seen triphones).
- Object in yellow/orange/red is "Event Vector"
  - Consider as a set of key-value pairs.
- The 0.01 is a variance floor (in here for C++ reasons, although shared among all stats..)

Fire a Linguist

- The late Fred Jelinek (founding director of the CLSP)
- Reported to have said, "Every time I fire a linguist, the error rate goes down."
- Apparently he insisted this had been taken out of context...
- Fred championed the "purely statistical" approach to speech recognition.
- In the same spirit, in Kaldi we avoid the use of "meaningful" hand-generated phonetic questions.

Clustering the phones

```
$ less steps/train_deltas.sh
.
<snip>
.
echo "Computing questions for tree clustering"

cat $lang/phones.txt | awk '{print $NF}' | grep -v -w 0 > $dir/phones.list
cluster-phones $dir/treeacc $dir/phones.list $dir/questions.txt 2> $dir/questions.log ||
exit 1;
```

- We cluster the phones to get questions.
- A question is just a set of phones.
- Would normally be a phonetic category.
- Here, just clusters based on acoustic similarity.
- Tree clustering -> hierarchy of sets of all sizes.

Looking at the questions

```
$ # line from steps/train_deltas.sh:
$ # scripts/int2sym.pl $lang/phones.txt < $dir/questions.txt > $dir/questions_syms.txt
$ less exp/tri1/questions_syms.txt
aa
aa ae aw ay eh ey f ow ts sil
aa ae aw ay eh ey ow
aa ae aw ay eh ow
aa aw ay ow
aa aw ow
ae
ae eh
```

- Some of the smaller sets look meaningful.
- Not all do, e.g. "aa" and "f" are not similar
- Questions in next stage (tri2a) are a bit better
- Presumably, monophone alignments are poor quality.

Clustering algorithm

- This clustering algorithm only used for a small part of the system (getting the questions)...
- but useful introduction to Kaldi's framework for clustering and decision trees.

- Abstract C++ interface "ClusterableInterface"
  - Represents some kind of stats and associated model type, from which we can get an objective function
  - Stats can be added together.

Clusterable interface

```
$ cd ~/kaldi-trunk/src
$ less itf/clusterable-interface.h
class Clusterable {
 public:
 <snip>
  virtual double Objf() const = 0;   // returns objective function
  virtual BaseFloat Normalizer() const = 0;  // typically returns data-count.
  virtual void Add(const Clusterable &other) = 0; // Adds other stats to this...
  virtual Clusterable *Copy() const; // Copy the object.
 <snip>
```

- Clustering routines act on generic objects satisfying this interface.
- In our stats, Objf() returns a Gaussian likelihood
- We'll split at the root to maximize the data likelihood, then split each branch...

Clusterable object

```
$ less tree/clusterable-classes.h
<snip>
class GaussClusterable: public Clusterable {
 public:
<snip>
  virtual void Add(const Clusterable &other_in);
  virtual void Sub(const Clusterable &other_in);
  virtual BaseFloat Normalizer() const { return count_; }
  virtual Clusterable *Copy() const;
<snip>
  double count_;
  Matrix<double> stats_; // two rows: sum, then sum-squared.
  double var_floor_;  // should be common for all objects created.
```

- GaussClusterable represents statistics for a Gaussian distribution.
- Contains count, sum and sum-squared of data (as in the tree-stats we saw).

Adding stats

```
$ less tree/clusterable-classes.cc
<snip>
void GaussClusterable::Add(const Clusterable &other_in) {
  assert(other_in.Type() == "gauss");
  const GaussClusterable *other =
      static_cast<const GaussClusterable*>(&other_in);
  count_ += other->count_;
  stats_.AddMat(1.0, other->stats_);
}
```

- the Add() function is very simple-- just add the stats together.

Getting likelihood

```
$ less tree/clusterable-classes.cc
<snip>
BaseFloat GaussClusterable::Objf() const {
  size_t dim = stats_.NumCols();
  Vector<BaseFloat> vars(dim);
  for (size_t d = 0; d < dim; d++) {
    double mean(stats_(0, d) / count_), var = stats_(1, d) / count_ - mean
        * mean;
    var = std::max(var, var_floor_);
    vars(d) = var;
  }
  BaseFloat ans = -0.5 * (vars.SumLog() + M_LOG_2PI * dim);
  return ans * count_;
}
```

- Compute the variance of the Gaussian
- Return the expected likelihood, times the count...
- Note: we apply a variance floor (otherwise the likelihood can go to infinity).

Clustering code

```
$ cd ~/kaldi-trunk/src
$ less bin/cluster-phones.cc
<snip>
      AutomaticallyObtainQuestions(stats,
                                   phone_sets,
                                   hmm_position_list,
                                   P,
                                   &phone_sets_out);
<snip>
```

- Note: phone_sets and phone_sets_out both of type vector<vector<int32> >
- phone_sets is just a single vector containing all the phones, in our example
  - A mechanism to let you keep some phone sets together through clustering.

Clustering code

```
$ cd ~/kaldi-trunk/src
$ less bin/cluster-phones.cc
<snip>
      AutomaticallyObtainQuestions(stats,
                                   phone_sets,
                                   pdf_id_list,
                                   P,
                                   &phone_sets_out);
<snip>
```

- stats is of type BuildTreeStatsType
  - vector of pair<EventVector, ClusterableInterface*>
  - EventVector specifies phone, context, etc.
- pdf_id_list is by default a vector containing just "1"... specifies to use only middle HMM-state's stats to cluster.

Clustering code

```
$ cd ~/kaldi-trunk/src
$ less bin/cluster-phones.cc
<snip>
      AutomaticallyObtainQuestions(stats,
                                   phone_sets,
                                   pdf_id_list,
                                   P,
                                   &phone_sets_out);
<snip>
```

- Note on coding style:
- Variables for function outputs are passed by pointer, and come after input parameters.
- Style guide (derived from Google style guide) dictates this.

Inside the clustering code

```
$ less tree/build-tree.cc
void AutomaticallyObtainQuestions( <snip>  ) {
 ...
  BuildTreeStatsType retained_stats;
  FilterStatsByKey(stats, kPdfClass, all_pdf_classes,
                   true,  // retain only the listed positions
                   &retained_stats);
 ...
}
```

- Looking at code of AutomaticallyObtainQuestions
- Call to FilterStatsByKey keeps only stats from HMM-state 1 (middle HMM-state)... configurable via all_pdf_classes variable.
- Note: kPdfClass is an enum that evaluates to -1... this is the EventMap "key" for "pdf-class" which is normally synonymous with HMM-state.

Inside the clustering code

```
$ less tree/build-tree.cc
void AutomaticallyObtainQuestions( <snip>  ) {
 ...
 ...
  std::vector<BuildTreeStatsType> split_stats;  // split by phone.
  SplitStatsByKey(retained_stats, P, &split_stats);
 ...
}
```

- Next statement splits stats up according to the central phone (monophone)
- Note: variable P (c.f. command-line option --central-position) is the center of context-window of phonemes.
- Value of P is 1 for triphone, 0 for monophone.

Inside the clustering code

```
$ less tree/build-tree.cc
void AutomaticallyObtainQuestions( <snip>  ) {
 ...
 ...
 ...
  std::vector<Clusterable*> summed_stats;  // summed up by phone.
  SumStatsVec(split_stats, &summed_stats);
 ...
}
```

- Next statement sums up all the stats for each phoneme (over all contexts, HMM-positions).
- Type of stats is now vector<Clusterable*>
  - I.e. we don't have the EventVector any more, that specifies context etc.
  - Phone is just index into vector.

Inside the clustering code

```
$ less tree/build-tree.cc
void AutomaticallyObtainQuestions( <snip>  ) {
 ...
 ...
 ...
  TreeClusterOptions topts;
  topts.kmeans_cfg.num_tries = 10;  // This is a slow-but-accurate setting,
  // we do it this way since there are typically few phones.

  std::vector<int32> assignments;  // assignment of phones to clusters. dim ==
summed_stats.size().
  std::vector<int32> clust_assignments;  // Parent of each cluster.  Dim == #clusters.
  int32 num_leaves;  // number of leaf-level clusters.
  TreeCluster(summed_stats_per_set,
              summed_stats_per_set.size(),  // max-#clust is all of the points.
              NULL,  // don't need the clusters out.
              &assignments,
              &clust_assignments,
              &num_leaves,
              topts);
 ...
}
```

- Here is the main call to tree-clustering routine.

Compiling the questions

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ less steps/train_deltas.sh
<snip>
compile-questions $lang/topo $dir/questions.txt $dir/questions.qst 2>$dir/compile_questions.log
<snip>

$
```

- Program "compile-questions" takes lists of phonemes...
- Transforms it into a C++ object (written to disk) that contains questions for each "key" in EventMap
- Sets up questions about HMM-state (0,1,2)...
- Here, some options can be set that affect tree-building.

Setting tree roots

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ less steps/train_deltas.sh
<snip>
# Have to make silence root not-shared because we will not split it.
scripts/make_roots.pl --separate $lang/phones.txt $silphonelist shared split \
    > $dir/roots.txt 2>$dir/roots.log || exit 1;

<snip>
```

- Set up sets of phones with "shared roots" for trees
- In this case, all phones have separate tree root
- If phones only differ in stress or tone etc., can be useful to share roots
  - This way, unseen variants still get a model.

Looking at roots file

```
$ less exp/tri1/roots.txt
not-shared not-split 48
shared split 33
shared split 32
shared split 21
shared split 7
shared split 26
shared split 17
```

- Note: integers correspond to phones; in general, can have lists of integers (to share roots).
- shared means HMM-states 0,1,2 share a root (can ask questions about HMM-state/pdf-id).
- split means we build a decision tree for this root (else, leave an un-split stub).

Building the decision tree

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ less steps/train_deltas.sh
<snip>
echo "Building tree"
build-tree --verbose=1 --max-leaves=$numleaves \
    $dir/treeacc $dir/roots.txt \
    $dir/questions.qst $lang/topo $dir/tree  2> $dir/train_tree.log
```

- Actually a set of decision trees (one per root)
- The max-leaves (e.g., 2000) is number of p.d.f's
- Some post-clustering done within each tree, after splitting.
  - This shares leaves, but only within each tree (e.g. per phone, not globally)

Building the decision tree

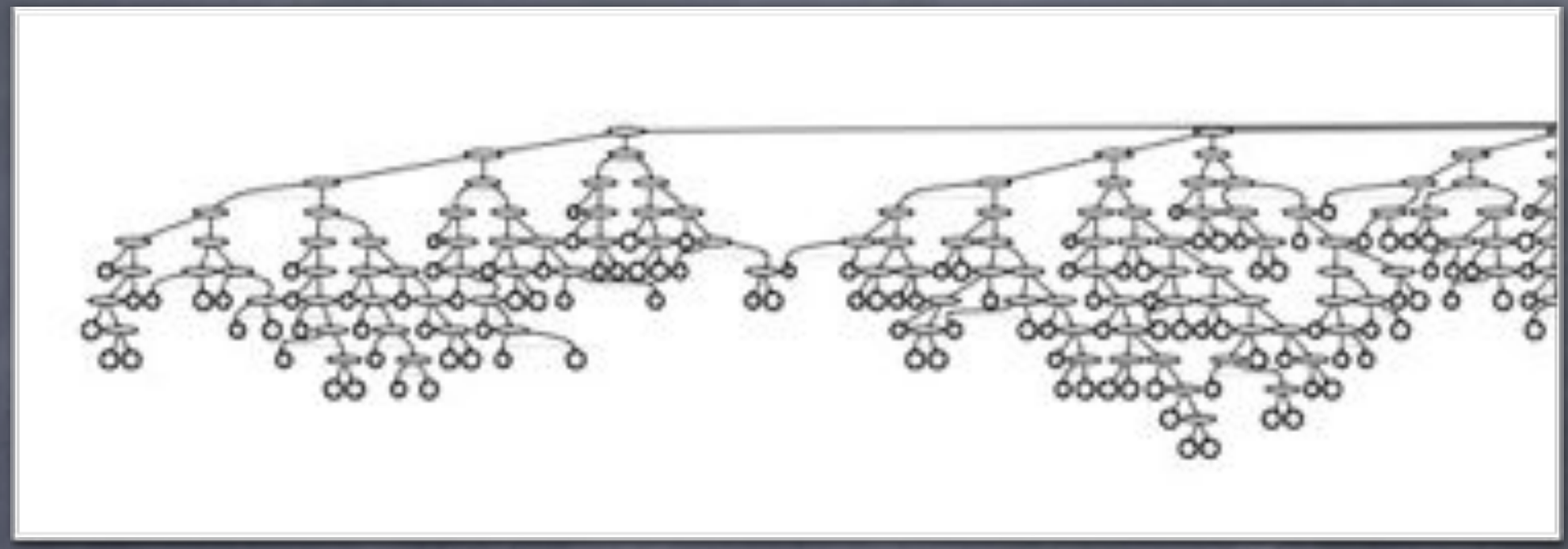
```
$ less exp/tri1/train_tree.log
build-tree --verbose=1 --max-leaves=1800 exp/tri1/treeacc exp/tri1/roots.txt exp/tri1/questions.qst data/
lang/topo exp/tri1/tree
Number of separate statistics is 19268
LOG (build-tree:BuildTree():build-tree.cc:162) BuildTree: before building trees, map has 52 leaves.
LOG (build-tree:SplitDecisionTree():build-tree-utils.cc:563) DoDecisionTreeSplit: split 1748 times, #leaves
now 1800
LOG (build-tree:BuildTree():build-tree.cc:188) Setting clustering threshold to smallest split 580.508
VLOG[1] (build-tree:BuildTree():build-tree.cc:197) After decision tree split, num-leaves = 1800, like-impr =
5.3741 per frame over 1.3679e+06 frames.
VLOG[1] (build-tree:BuildTree():build-tree.cc:201) Including just phones that were split, improvement is
6.13816 per frame over 1.19763e+06 frames.
LOG (build-tree:BuildTree():build-tree.cc:216) BuildTree: removed 378 leaves.
VLOG[1] (build-tree:BuildTree():build-tree.cc:223) Objf change due to clustering -0.103276 per frame.
VLOG[1] (build-tree:BuildTree():build-tree.cc:226) Normalizing over only split phones, this is: -0.11796 per
frame.
VLOG[1] (build-tree:BuildTree():build-tree.cc:229) Num-leaves is now 1422
<snip>
VLOG[1] (build-tree:main():build-tree.cc:160) For pdf-id 1119, low count 99
LOG (build-tree:main():build-tree.cc:208) Wrote tree
```

- Likelihood improvement from tree splitting is important diagnostic.
- More likelihood improvement is generally better (means context is helping more).

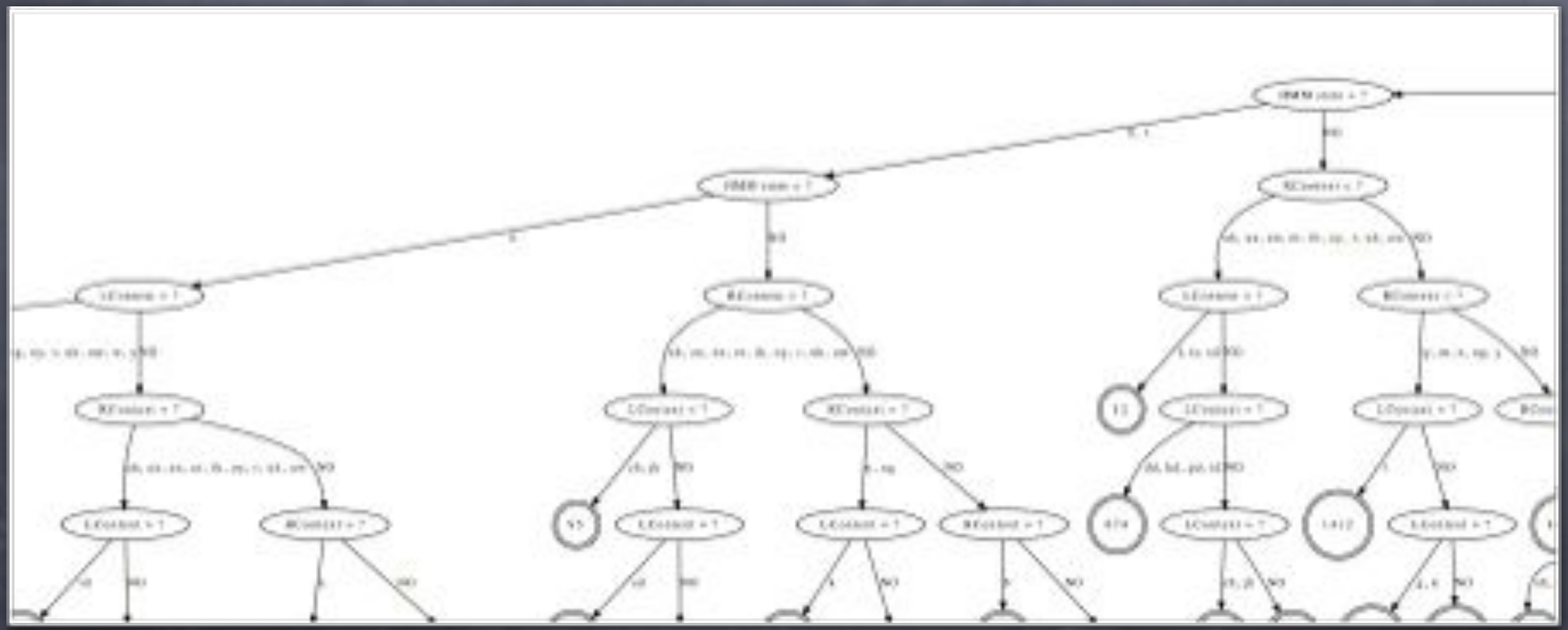
Drawing the decision tree

```
$ . path.sh
$ draw-tree data/lang/phones.txt exp/tri1/tree | dot -Tps -Gsize=8,10.5 | ps2pdf - ~/tree.pdf
```

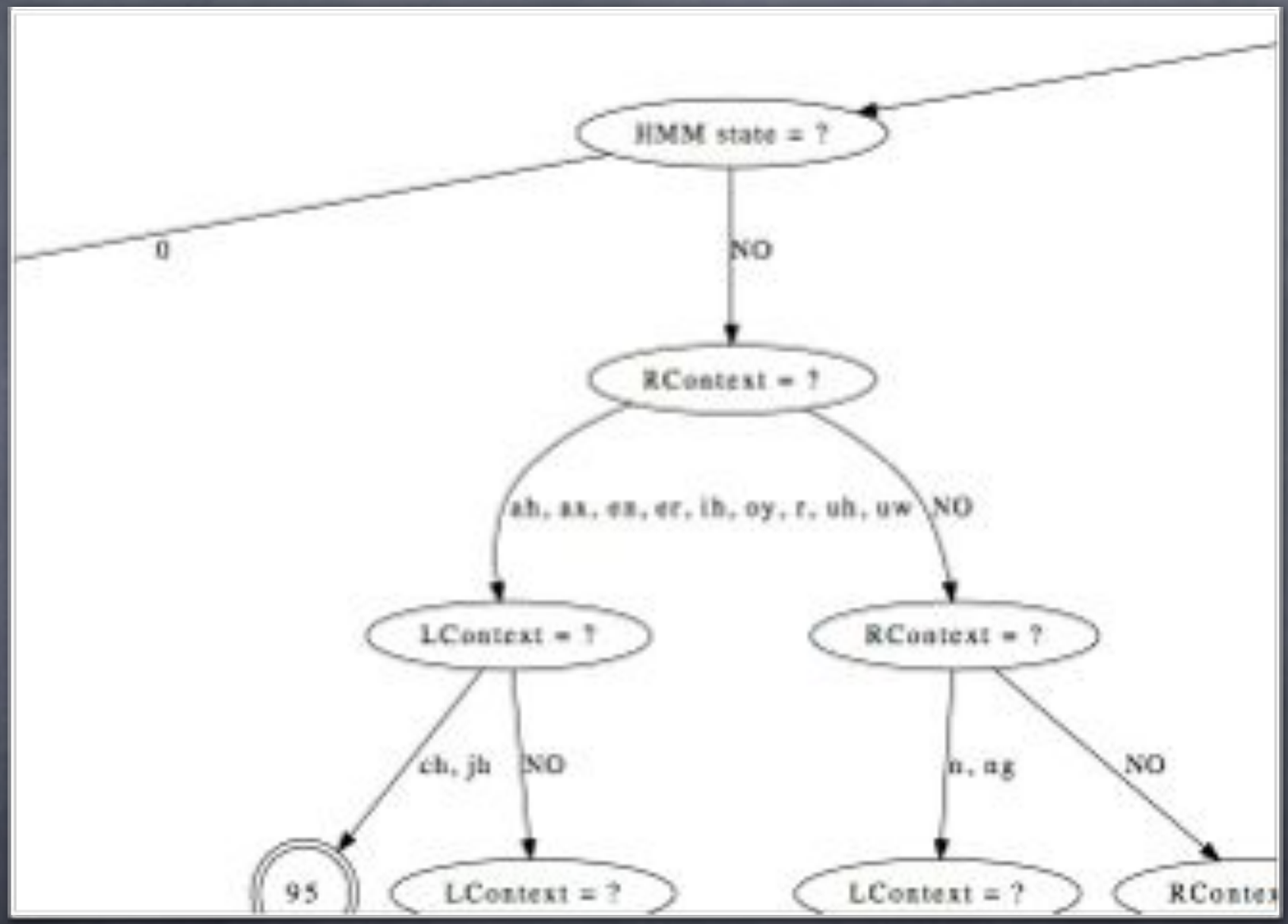
Drawing the decision tree



Drawing the decision tree



Drawing the decision tree



Decision tree leaves

- Decision tree leaves are integers (no names!)
- We call these "pdf-ids".
- They are zero-based (i.e. numbered from zero)
- Caution: some integer identifiers in Kaldi are one-based (e.g. "transition-ids")
- This is for compatibility with OpenFst, where zero is "special".
- Where possible we prefer zero-based indexing.

Decision tree -- differences from "standard" approach

- Can share tree roots among phones
- Same tree root for all the HMM-states of a phone (or phone-set)
  - Ask questions about HMM-state.
- Automatically obtained questions
- Leaves post-clustered after tree splitting

Decision tree object

```
$ less tree/context-dep.h
<snip>
class ContextDependency: public ContextDependencyInterface {
 public:
  virtual int32 ContextWidth() const { return N_; }
  virtual int32 CentralPosition() const { return P_; }

  /// returns success or failure; outputs pdf to pdf_id
  virtual bool Compute(const std::vector<int32> &phoneseq, int32 pdf_class, int32 *pdf_id) const;
...
```

- Decision tree object: type "ContextDependency"
- Written to file called "tree"
- Maps from [window of phones, HMM-state] to integer pdf-id.
- E.g. (aa/n/d, 3) -> 1402

Initializing the model

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ less steps/train_deltas.sh
<snip>
gmm-init-model  --write-occs=$dir/1.occs  \
    $dir/tree $dir/treeacc $lang/topo $dir/1.mdl 2> $dir/init_model.log
```

- This program reads the tree, tree accumulators, and topology.
- It outputs the model file 1.mdl

Initializing the model

```
$ cd ~/kaldi-trunk/src
$ less gmmbin/gmm-init-model.cc
<snip>
    {
      Output ko(model_out_filename, binary);
      trans_model.Write(ko.Stream(), binary);
      am_gmm.Write(ko.Stream(), binary);
    }
    KALDI_LOG << "Wrote tree and model.";
```

- Write two objects to the model file.
- "trans_model" (type: TransitionModel)
- "am_gmm" (type: AmDiagGmm)
- Some programs that read the model file, only read the TransitionModel object.
- This makes them model-type independent.

Initializing the model

```
$ cd ~/kaldi-trunk/src
$ less gmmbin/gmm-init-model.cc
<snip>
    {
      Output ko(model_out_filename, binary);
      trans_model.Write(ko.Stream(), binary);
      am_gmm.Write(ko.Stream(), binary);
    }
    KALDI_LOG << "Wrote tree and model.";
```

- Note: "Output" object opens a generalized filename (works with files, stdin/stdout, piped commands)
- The Write functions of Kaldi objects take an ostream, and a bool (for binary/text mode)
- Idea is to make code easily refactorable (if they took the "Output" object, too Kaldi-dependent.

GMM code

```
$ less gmm/diag-gmm.h
<snip>
class DiagGmm {
 friend class DiagGmmNormal;
<snip>
  /// Returns the log-likelihood of a data point (vector) given the GMM
  BaseFloat LogLikelihood(const VectorBase<BaseFloat> &data) const;
<snip>
 private:
  /// Equals log(weight) - 0.5 * (log det(var) + mean*mean*inv(var))
  Vector<BaseFloat> gconsts_;
  bool valid_gconsts_;   ///< Recompute gconsts_ if false
  Vector<BaseFloat> weights_;         ///< weights (not log).
  Matrix<BaseFloat> inv_vars_;        ///< Inverted (diagonal) variances
  Matrix<BaseFloat> means_invvars_;  ///< Means times inverted variance
```

- Object DiagGmm is a single Gaussian Mixture Model
- Parameters stored in "exponential model" form for fast likelihood evaluation
- Convert to DiagGmmNormal for easy updates etc. (this stores them more conventionally).

GMM model set

```
$ less gmm/am-diag-gmm.h
<snip>
class AmDiagGmm {
<snip>
 private:
  std::vector<DiagGmm*> densities_;
<snip>
```

- Object AmDiagGmm contains a vector of DiagGmm
- Indexed by "pdf-id" (remember, this is a zero-based integer index).
- This object knows nothing about transitions, topology, etc.

Transition model

```
$ less hmm/transition-model.h
<snip>
class TransitionModel {

 public:
  /// Initialize the object [e.g. at the start of training].
  TransitionModel(const ContextDependency &ctx_dep,
                  const HmmTopology &hmm_topo);

  void Read(std::istream &is, bool binary);  // note, no symbol table: topo object always read/written w/o
symbols.
  void Write(std::ostream &os, bool binary) const;
<snip>
  // Transition-parameter-getting functions:
  BaseFloat GetTransitionProb(int32 trans_id) const;
```

- Object TransitionModel responsible for storing HMM transition probabilities
- Also keeps track of HMM topologies (contains the HmmTopology object)
- Defines "transition-ids"-- an index that corresponds with transition probs, and useful for other reasons.

Transition parameters

- The decision-trees are at the individual state level, not the whole-HMM level.
- Therefore there may be many more HMMs than decision-tree leaves (combinatorial explosion)
- We tie the transition-model parameters the same way as the decision-tree leaves
  - although if the monophone/center phone is different, we have a different transition-prob.
- Transitions **out of** a given state are tied like that state.

End of this lecture