Speech REcognition

- a practical guide

In this lecture:

- Overview of the course
- Getting started
- Speech feature extraction

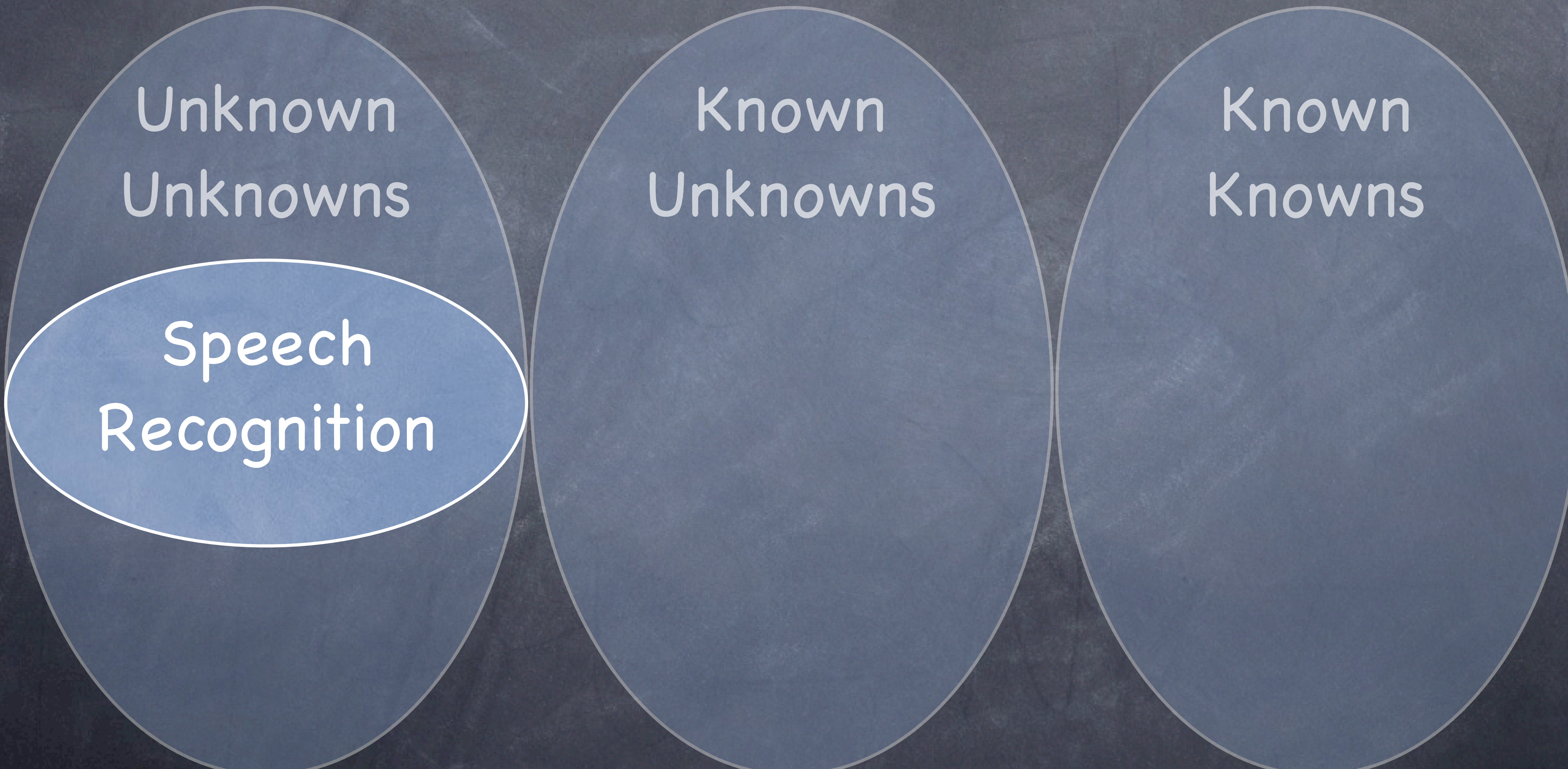
Overview of the course

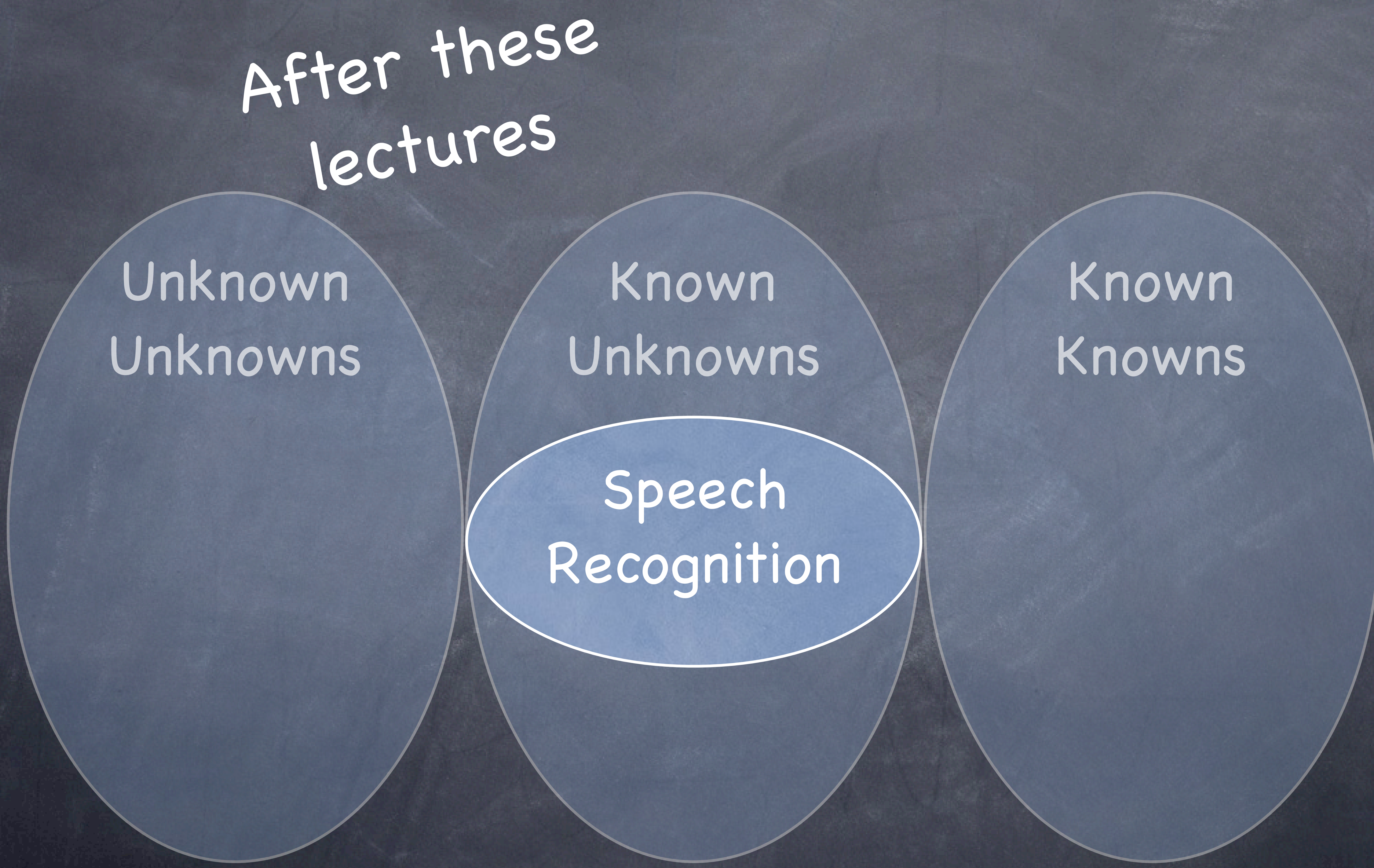
Unknown Unknowns

Known Unknowns

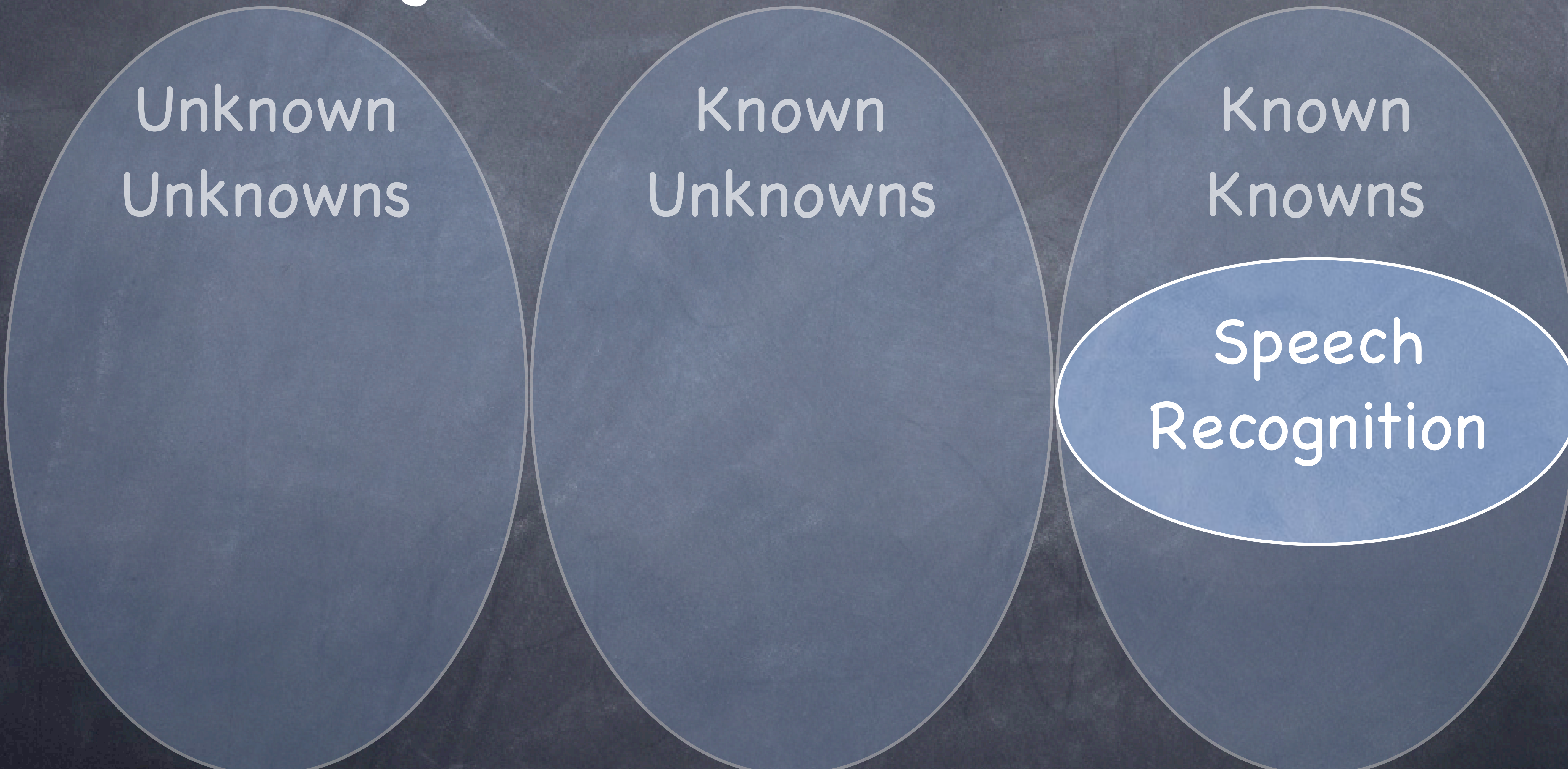
Known Knowns

Now


Unknown Unknowns
Speech Recognition
Known Unknowns
Known Knowns


After these lectures
Unknown Unknowns
Known Unknowns
Speech Recognition
Known Knowns

After further study



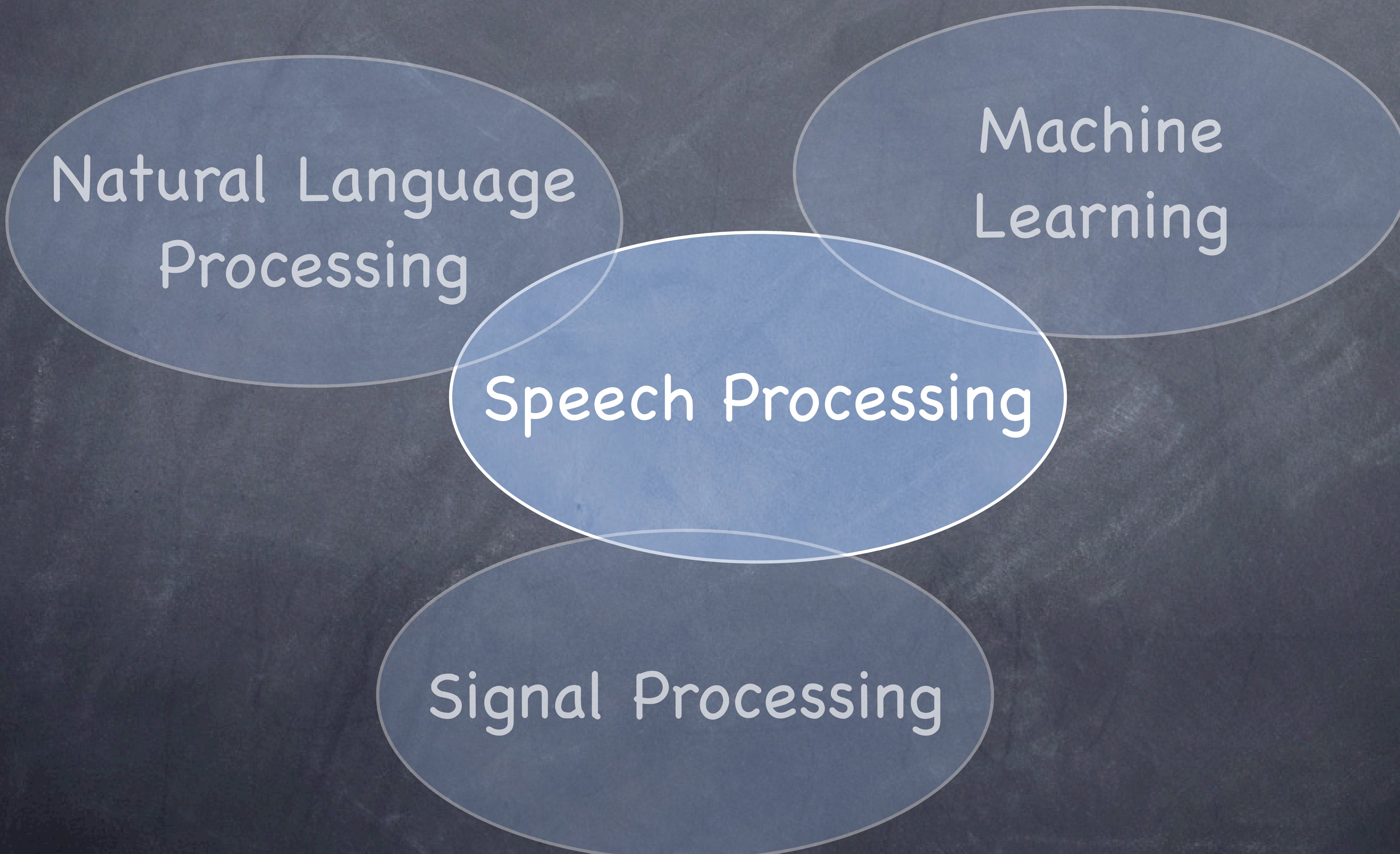
Structure of this lecture series

- A series of 45-minute lectures
- Each one will combine:
  - Some of the theory of speech recognition
  - Practical examples with the Kaldi toolkit
    - Note: various toolkits exist.
    - speech recognition toolkit
    - I believe Kaldi is the best one... but I wrote much of it.
    - Note: this was released ~1 year ago.

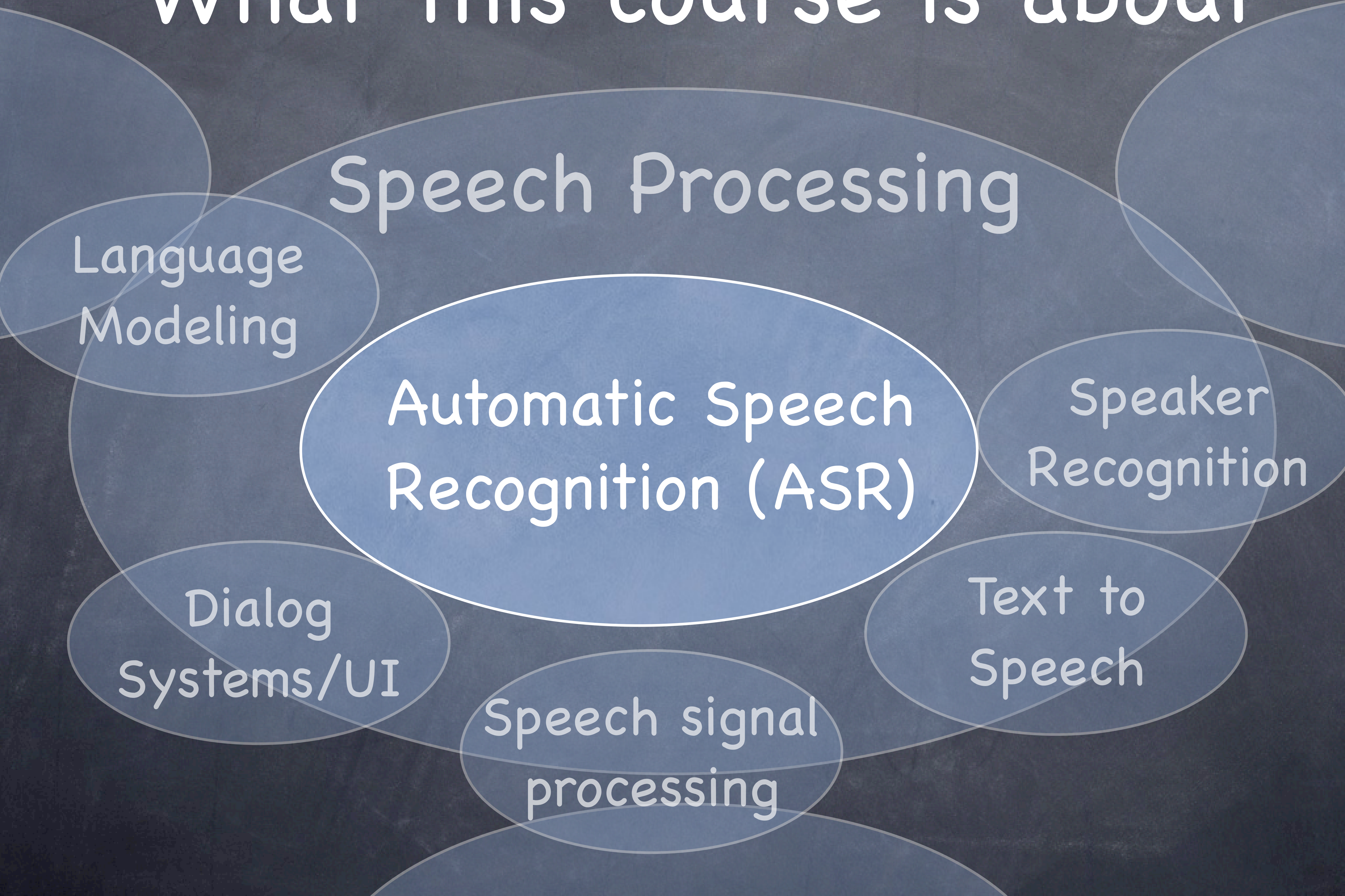
Prerequisites

- It will be helpful if you have encountered:
  - Statistical models
  - UNIX shell scripts
  - C++
- If a section requires background knowledge of some kind, we will suggest search terms.
  - e.g.: bash scripting

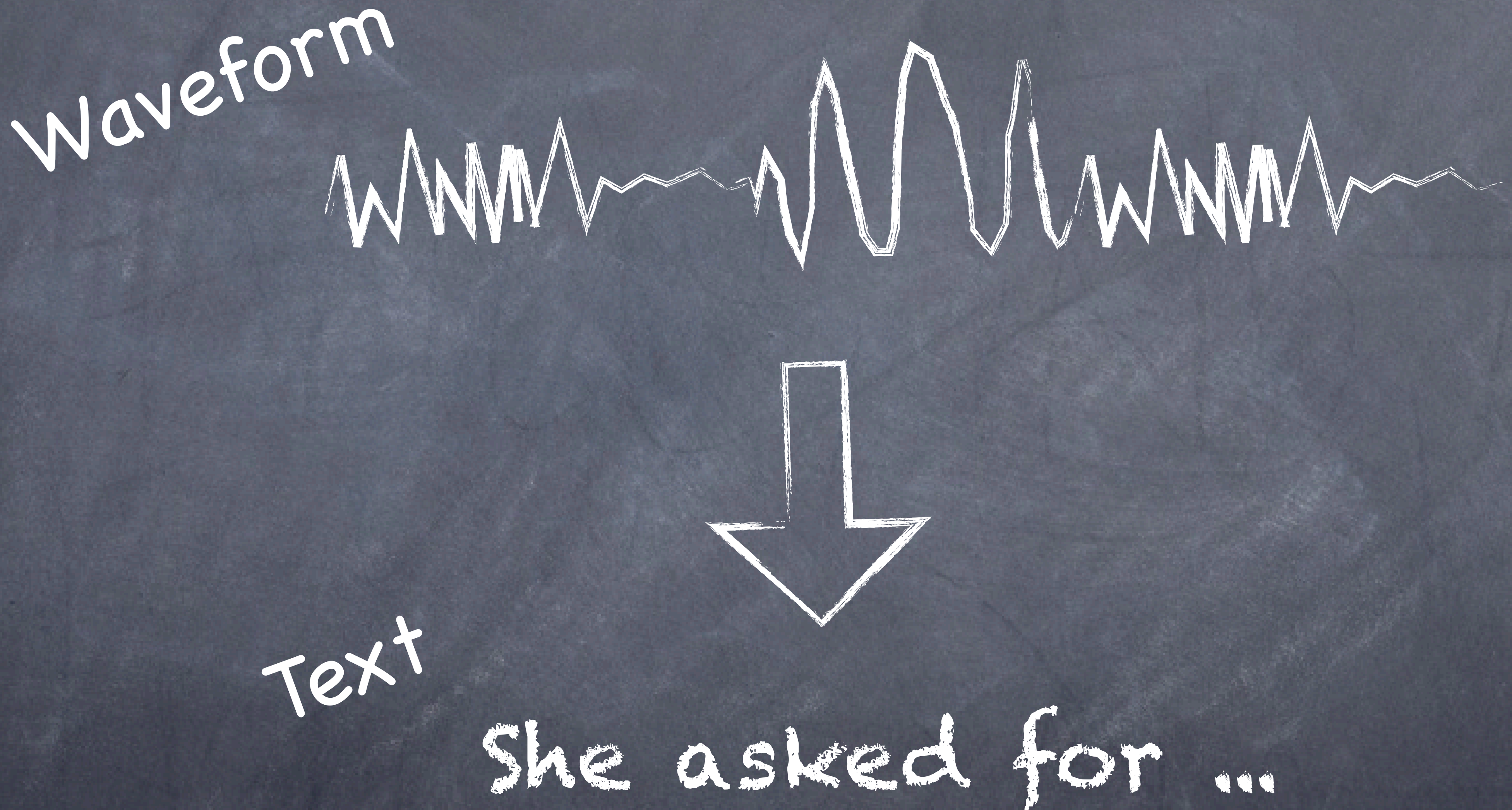
What this course is about



What this course is about



What is Speech Recognition?



How we do it

- Given "training data" from the target language, we'll train a statistical model of speech.

statistical model

- This model will assign probabilities to (some sentence) producing (some waveform)
- Given a waveform, we can work out the most likely sentence.
- This won't be guaranteed accurate.

Data resources required

- A labeled corpus
  - i.e. a collection of recordings of speech
  - a record of what was spoken for each one
- A pronouncing dictionary, a.k.a. “lexicon”
  - Says, for each word, what the sequence of “phonemes” (speech sounds) is.
  - Not necessary in phonetically written languages
- Possibly some extra text to train “language model”

Finding speech data

- A lot of speech resources are available from the Linguistic Data Consortium (LDC)
- Also Appen, ELRA
- None of this is for free. Typically one to several thousand dollars for LDC databases
  - Not a download. It's FedEx.
- Some lexicons available for free (e.g. CMUDict)
- A limited amount of free speech data is available.

gutenberg audio

Other Resources

- To do large-scale speech training (on hundreds of hours of data), would also need:
  - A cluster of machines (at least 20 or so cores in total, preferably more), running e.g. GridEngine
  - A few hundred gigabytes of space on a fast disk (e.g. NFS mounted)
  - Fast local network

What you will be able to do

- If you listen to and understand this lecture series, you should be able to:
  - build and (somewhat) understand a command-line speech recognition system
- You will **not** be able to:
  - build a dialog system or speech user interface
  - get perfect accuracy (50-95% is normal range, except for yes/no/digit type dialogs)

How to follow these lectures

- I will be describing how to run the Kaldi software
- Better to watch or attend the lecture without taking notes
- Slides and video will be made available (follow links from kaldi.sf.net)
- For running the examples, do it after the lecture (get the commands from the slides)

Getting started

What you need

- Some kind of UNIX-based system (Linux, Mac, cygwin should all work).
- Plenty of memory (e.g. 5G), disk space (e.g. 20G).
- Fast Web connection, or LDC data on your system.
- You may need to install some packages
  - e.g. subversion (svn), wget, g++
  - System-dependent: figure it out yourself or ask your sysadmin.

Installing Kaldi

```
$ ## see instructions at http://kaldi.sf.net
$ ## first cd to somewhere with a lot of space.
$ svn co https://kaldi.svn.sourceforge.net/svnroot/kaldi/trunk kaldi-trunk
$ cd kaldi-trunk/tools
$ ./install.sh  ## Installs some stuff Kaldi depends on... takes a while
$ cd ../src
$ ./configure
$ make -j 8   ## -j 8 makes with 8 jobs in parallel; should not
$             ## exceed number of cores on your machine.
```

- If that worked, congratulations.
- Otherwise, try to figure out what went wrong.
- Look carefully at the output of steps that failed.

How to get help

- If any step in this course doesn't run..
- Check for obvious stuff like programs that are invoked but not installed.
- Ask at kaldi-developers@lists.sourceforge.net
- Please, no non-Kaldi questions, e.g. how do I change directories, how do I install awk.
- If you fix something, contact us.

What we installed (1)

```
$ cd ~/kaldi-trunk  # assuming it was in your homedir
$ ls
COPYING  INSTALL  README.txt  egs  misc  src  tools  windows
$ # Note: "tools/", "src/" and "egs/" are most important.
$ ls tools/
ATLAS        interpolatedwrite-5.60.02.patch  openfst.patch
CLAPACK_include    irstlm          INSTALL         atlas3.8.3.tar.gz
sctk-2.4.0  openfst            sctk-2.4.0-20091110-0958.tar.bz2
install.sh     openfst-1.2.10         sph2pipe_v2.5
install_atlas.sh   openfst-1.2.10.tar.gz      sph2pipe_v2.5.tar.gz
```

- Various tools Kaldi depends on.
- OpenFst: Weighted Finite State Transducer library
- ATLAS/CLAPACK: standard linear algebra libraries
- "scoring", audio format conversion tools....

What we installed (2)

```
$ cd ~/kaldi-trunk  # assuming it was in your homedir
$ cd src
$ ls
Doxyfile configure   fstext      lat       nnet_cpu tied
INSTALL     decoder     gmm       latbin       nnetbin     tiedbin
Makefile doc       gmmbin       lm    nnetbin_cpu transform
NOTES    feat     hmm       machine-type   optimization   tree
TODO     featbin     itf       makefiles   rnn      util
base     fgmmbin     kaldi.mk matrix      sgmm
bin      fstbin      kaldi.mk.bak   nnet     sgmmbin
```

- Mostly directories containing code.
- Those ending in bin/ contain Kaldi programs
  - There are a large number of programs, each with a fairly simple function.

Running the examples

```
$ cd ~/kaldi-trunk  # assuming it was in your homedir
$ cd egs
$ ls
README.txt  gp rm  swbd  timit  wsj
$ cd rm
$ ls
README.txt  s1 s2  s3 s4
$ cd s3  # The s3 example scripts are the most normal one.
$ ls
RESULTS  conf  data  exp  local  path.sh  run.sh  scripts  steps
```

- There are example scripts for various data-sets.
- We'll use Resource Management (smallest one).
- Very easy task: clean, planned speech, small vocabulary. (Spoken commands to computer).

Finding the data

```
$ cd ~/kaldi-trunk/egs/rm
$ cat README.txt
About the Resource Management corpus:
    Clean speech in a medium-vocabulary task consisting
    of commands to a (presumably imaginary) computer system.  About 3
    hours of training data.
    Available from the LDC as catalog number LDC93S3A (it may be
    possible to get the same data using combinations of other catalog
    numbers, but this is the one we used).
```

- See if you have this data on your system
- It's $1000 from LDC if non-member.
- Look for directory containing subdirs:

  rm1_audio1 rm1_audio2 rm2_audio

If you don't have the data

- If your institution is not an LDC member and doesn't want to pay for the data:
    - you can use the scripts in rm/s4
    - Uses precomputed features derived from a subset of the RM data
    - Will be downloaded from the Internet.

Thanks to Vassil Panayotov for contributing this recipe.

Looking at the data

```
$ find /export/corpora5/LDC/LDC93S3A/rm_comp  | head
/export/corpora5/LDC/LDC93S3A/rm_comp
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio/3-2.2
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio/3-2.2/rm2
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio/3-2.2/rm2/ex_train
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio/3-2.2/rm2/ex_train/lpn0_7
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio/3-2.2/rm2/ex_train/lpn0_7/tc1125.wav
/export/corpora5/LDC/LDC93S3A/rm_comp/rm2_audio/3-2.2/rm2/ex_train/lpn0_7/tc0966.wav
$ less /export/corpora5/LDC/LDC93S3A/rm_comp/rm1_audio1/rm1/doc/al_sents.txt
; al_sents.txt - updated 09/20/89
<snip>
What is the constellation's gross displacement in long tons? (SR001)
Is Ranger's earliest CASREP rated worse than hers? (SR002)
Show me all alerts. (SR003)
Give Bainbridge's CASREPs from the last 7 months. (SR004)
Show the Enterprise's home port. (SR005)
Draw Texas's last 3 H.F.D.F. sensor posits. (SR006)
```

- Note: .wav files are not really .wav, they are .sph
- Use tools/sph2pipe_v2.5/sph2pipe to convert

sphere format

The word-pair grammar

```
$ less /export/corpora5/LDC/LDC93S3A/rm_comp/rm1_audio1/rm1/doc/wp_gram.txt
/*
*****************************************************************************
 *                COPYRIGHT 1987.  BBN LABORATORIES, INCORPORATED
 *
 *                                ALL RIGHTS RESERVED
****************************************************************************
 * File: patts_snor_word_pair.text
 *
 * This file contains a specification for the 'word-pair' grammar developed
 * at BBN.
 * The grammar allows all two word sequences (bigrams) possible in the DARPA
 * continuous speech resource management database as defined by the sentence
 * pattern grammar.
...
```

- The RM database comes with a "word-pair grammar"
- For the other Kaldi examples, we use statistical language models.

n-gram model

Bayes' rule and ASR

$$P(S \mid audio) = \frac{p(audio \mid S)\;\; P(S)}{p(audio)}$$

Note:
p() = likelihood
P() = probability

- Here, S is the sequence of words, P(S) is language model, e.g. n-gram model or probabilistic grammar.
- p(audio | S) is a sentence-dependent statistical model of audio production, trained from data.
- Given a test utterance, we pick S to maximize P(S | audio). I.e. the most likely sentence.
- Note: p(audio) is a normalizer that doesn't matter.

Preparing the data

```
$ cd ~/kaldi-trunk/egs/rm/s3
$  ## we're running the steps from run.sh ##
$ local/rm_data_prep.sh /export/corpora5/LDC/LDC93S3A/rm_comp
$ local/rm_format_data.sh
$ ls data
lang  lang_test  local  test_feb89  test_feb91  test_mar87  test_oct87
test_oct89  test_sep92  train
```

- Putting data in form that Kaldi scripts understand.
- data/lang contains language-specific stuff (also see data/lang_test which contains the grammar too).
- data/train contains training data (data/test_feb89 etc. have same format)

Language-specific stuff

```
$ head -5 data/lang/phones.txt
<eps> 0
aa 1
ae 2
ah 3
ao 4
aw 5
$ head -2 data/lang/words.txt
head -4 data/lang/words.txt
<eps> 0
A  1
A42128   2
AAW   3
$ cat data/lang/silphones.csl
48
$  ## Note: just one silence phone in this setup.
```

- *.txt are symbol tables in OpenFst format
- Map between strings and ints; Kaldi code uses ints.

The lexicon

```
$ fstprint --isymbols=data/lang/phones.txt --osymbols=data/lang/words.txt
data/lang/L.fst  | head
0  1  <eps> <eps> 0.693147182
0  1  sil   <eps> 0.693147182
1  1  ax A  0.693147182
1  2  ax A  0.693147182
1  3  ey A42128
1  15 ey AAW
1  21 ae ABERDEEN
1  26 ax ABOARD
1  30 ax ABOVE
```

- The lexicon (pronouncing dictionary) is in binary OpenFst format
- Can view it as text using the command above.

Weighted Finite State Transducers (WFSTs)

- Various resources for learning WFSTs, OpenFst
  - Informal intro by me to WFSTs (read slides first)
    - http://old-site.clsp.jhu.edu/news-events/abstract.php?sid=20110902
  - More formal one, search for hbka.pdf
  - Paul Dixon tutorial: apsipa_09_tutorial_dixon_furui.pdf
  - For OpenFst resources/tutorial: www.openfst.org
  - Next slides: very quick intro.

WFST quick intro: FSAs

- Finite State acceptor (FSA) is a finite representation of a possibly infinite set of strings.
  - Has a finite #states. One is "initial state". States can be labeled "final".
  - Arcs between states have symbols on them (or special symbol epsilon meaning no symbol)
  - String == symbol-sequence.
  - String accepted if there's a path with that symbol-sequence on, from initial->final state.

WFST quick intro: WFSAs

- WFSA is like FSA but adding costs to the transitions and final-states.
- String "accepted" with weight determined by minumum-cost path from initial->final.
- The notion of cost can be generalized.
- We call them "weights". Operations + and *, satisfying axioms of a "semiring"
    - A weight is "multiplied" along paths, "added" across paths.

WFST quick intro: FSTs

- Finite State transducer (FST) is (from the point of view of its name) is an object that "transduces" (converts) one string into another.
- Like FSA but two symbols on each arc: "input" and "output".
  - Mathematically, represents a set of pairs of strings: (input-string, output-string).
  - "transducer" name is a bit misleading.
  - Notion of "composition" (like function composition)

WFST quick intro: WFSTs

- WFST combines the two-symbol idea of FSTs, with the weighting idea of FSAs.
- Keywords:
  - Determinization, minimization, composition
  - equivalent, epsilon-free, functional
  - on-demand algorithm
  - weight-pushing, epsilon removal
- You might want to find out what these mean.

Data directory format

```
$ ls data/train  ## note: it would look like this after the next step.
spk2gender  spk2utt  text  utt2spk  wav.scp
$ head -2 data/train/wav.scp
trn_adg04_sr009 sph2pipe -f wav /foo/rm1_audio1/rm1/ind_trn/adg0_4/sr009.sph |
trn_adg04_sr049 sph2pipe -f wav /foo/rm1_audio1/rm1/ind_trn/adg0_4/sr049.sph |
$ head -2 data/train/text
trn_adg04_sr009 SHOW THE GRIDLEY+S TRACK IN BRIGHT ORANGE
trn_adg04_sr049 IS DIXON+S LENGTH GREATER THAN THAT OF RANGER
$ head -2 data/train/utt2spk
trn_adg04_sr009 adg0
trn_adg04_sr049 adg0
```

- Most of these files map from utterance-id to (something)
- Kaldi "Table" concept: collection of objects indexed by a string.

The Table concept

- A Table is a collection of objects indexed by a string (string must be nonempty, space-free).
  - E.g. a collection of matrices indexed by utterance-id, representing features.
- "Templates" in C++: e.g. vector<int> is a vector of integers. Mechanism for generic code.
- The basic concept is: Table<Object>, e.g. Table<int>, Table<Matrix<float> >
- Handles access to objects on disk (or pipes, etc.)

Tables: form on disk

- Two ways objects are stored on disk:
  - "scp" (script) mechanism: .scp file specifies mapping from key (the string) to filename or pipe:

```
$ head -2 data/train/wav.scp
trn_adg04_sr009 sph2pipe -f wav /foo/rm1_audio1/rm1/ind_trn/adg0_4/sr009.sph |
trn_adg04_sr049 sph2pipe -f wav /foo/rm1_audio1/rm1/ind_trn/adg0_4/sr049.sph |
```

  - "ark" (archive) mechanism: data is all in one file, with utterance id's (example below is in text mode):

```
$ head -2 data/train/text
trn_adg04_sr009 SHOW THE GRIDLEY+S TRACK IN BRIGHT ORANGE
trn_adg04_sr049 IS DIXON+S LENGTH GREATER THAN THAT OF RANGER
```

Specifying Tables on command line

- Strings passed from command line say how to read or write Tables.
  - Note: the type of object expected, and whether to read or write, is determined by the program itself.
  - A string interpreted as specifying how to write a Table, we call a "wspecifier" in code, etc.
  - A string that specifies how to read a Table is called an "rspecifier".

Examples of writing Tables

wspecifier	meaning
ark:foo.ark	Write to archive "foo.ark"
scp:foo.scp	Write to files using mapping in foo.scp
ark:-	Write archive to stdout
ark,t:|gzip -c >foo.gz	Write text-form archive to foo.gz
ark,t:-	Write text-form archive to stdout
ark,scp:foo.ark,foo.scp	Write archive and scp file (see below)

- Last one is a special case: write archive, and .scp file specifying offsets into that archive (for efficient random access). Here, .scp file is like an index.

Examples of reading Tables

rspecifier	meaning
ark:foo.ark	Read from archive foo.ark
scp:foo.scp	Read as specified in foo.scp
ark:-	Read archive from stdin
ark:gunzip -c foo.gz|	Read archive from foo.gz
ark,s,cs:-	Read archive (sorted) from stdin...

- In last one, "s" asserts archive is sorted, "cs" asserts it will be called in sorted order.
  - Allows memory-efficient random access on archive.

C++ level Table code

- Note: there is actually no Table<Object> class.
- There are three: SequentialTableReader, RandomAccessTableReader, and TableWriter.

```
SequentialTableReader<Matrix<float> > mat1_reader(rspecifier1);
RandomAccessTableReader<Matrix<float> > mat2_reader(rspecifier2);
TableWrite<Matrix<float> > mat_writer(wspecifier);
for (; !mat1_reader.Done(); mat1_reader.Next()) {
  const Matrix<float> mat1(mat1_reader.Value());
  std::string key = mat1_reader.Key();
  if (mat2_reader.HasKey(key)) {
    Matrix<float> mat2(mat2_reader.Value());
    Matrix<float> prod(mat1.NumRows(), mat2.NumCols());
    prod.AddMatMat(1.0, mat1, kNoTrans, mat2, kNoTrans);
    mat_writer.Write(key, prod);
  }
}
```

Shell level Table example

- This fake example imagines the code on the previous slide was in a program called multiply-matrices.
- In reality, Kaldi programs are a little higher level than this (although there is a program "transform-feats" that does this as a special case).

```
$ multiply-matrices "scp:feats.scp" \
  "ark:gunzip –c transforms.gz|" \
  "ark,t:|gzip –c >transformed_feats.gz"
$
```

Feature processing

Speech audio processing

- The most useful information in speech is frequency domain
  - e.g. position of peaks in amplitude called "formants" that vary between vowels
  - We use short-time Fourier spectrum
  - Further process this to reduce dimension and make it more Gaussian distributed.

gaussian distribution

Audio processing (simple version)

- Input is 16kHz sampled audio.
- Take a 25ms window (shift by 10 ms each time; we will output a sequence of vectors, one every 10ms)
- Multiply by windowing function e.g. Hamming

  Hamming window
- Do fourier transform

  FFT
- Take log energy in each frequency bin
- Do discrete cosine transform (DCT): (gives us the "cepstrum")

  cepstrum
- Keep the first 13 coefficients of the cepstrum.

Audio processing (details)

- Pre-scale the frequency axis with “mel” (perceptual) scale before doing DCT

mel scale

- Don’t take DCT of individual frequency components: average energy over triangular “bins”, equally spaced in mel scale
- “Pre-emphasize” signal (do $s'(t) = s(t) - 0.97\, s(t-1)$) ... reduces aliasing artifacts w/ Hamming (?)
- Add a little noise to signal: “dithering”--> no log(0)
- Result is MFCC (Mel Frequency Cepstral Coeffs.)
  - Kaldi also supports “PLP” (perceptual linear prediction)-- usually a bit better.

Audio processing (script)

```
## assumes your shell is bash.  Uses 4 cpus (parameter 4)
featdir=mfcc_feats  ## Note: put this somewhere with disk space

for x in train test_mar87 test_oct87 test_feb89 test_oct89 \
     test_feb91 test_sep92; do
  steps/make_mfcc.sh data/$x exp/make_mfcc/$x $featdir 4
  #steps/make_plp.sh data/$x exp/make_plp/$x $featdir 4
done
```

- For training set and each of the test sets, make the features with 4 CPUs (on local machine).
  - Puts features e.g. in data/train/feats.scp

```
head data/train/feats.scp
trn_adg04_sr009 /home/dpovey/data/kaldi_rm_feats/raw_mfcc_train.1.ark:16
trn_adg04_sr049 /home/dpovey/data/kaldi_rm_feats/raw_mfcc_train.1.ark:23395
trn_adg04_sr089 /home/dpovey/data/kaldi_rm_feats/raw_mfcc_train.1.ark:37310
```

Audio processing (script)

- Main command run by steps/make_mfcc.sh:

```
$ head -1 exp/make_mfcc/train/make_mfcc.1.log
compute-mfcc-feats --verbose=2 --config=conf/mfcc.conf \
  scp:exp/make_mfcc/train/wav1.scp \
  ark,scp:/data/mfcc/raw_mfcc_train.1.ark,/data/mfcc/raw_mfcc_train.1.scp
```

- First argument "scp:..." tells it to find filenames (actually commands) in [dir]/wav1.scp
- Second argument "ark,scp:..." tells it to write an archive, and an index into the archive.
- Archive contains (num-frames)x13 matrix of features, for each utterance.

Audio processing (code)

```
## simplified extract from src/featbin/compute-mfcc-feats.cc

main(int argc, char *argv[]) {
  // <snip>: parse command line arguments.
  Mfcc mfcc(mfcc_opts);

  SequentialTableReader<WaveHolder> reader(wav_rspecifier);
  BaseFloatMatrixWriter writer(feat_wspecifier);  // note: a typedef.
  for (; !reader.Done(); reader.Next()) {
    string utt = reader.Key();
    const WaveData &wave_data = reader.Value();
    int32 channel = 0; # Let's assume mono data for now.
    BaseFloat vtln_warp = 1.0; # Gloss over VTLN (vocal tract len. norm.)
    SubVector<BaseFloat> waveform(wave_data.Data(), this_chan);
    Matrix<BaseFloat> features;
    mfcc.Compute(waveform, vtln_warp, &features, NULL);
    writer.Write(utt, features);
  }
}
```

Note on Tables

- We said Table types were templated on the type they store, e.g. TableWriter<Matrix<float> >
- This is a simplification: we actually template on a "Holder" type that tells the Table code how to read and write the object.
- Necessary because objects don't have uniform read/write methods. (must work for fundamental types)

Audio processing (code)

```
## simplified extract from src/feat/feature-mfcc.cc
void Mfcc::Compute(const VectorBase<BaseFloat> &wave,
                   Matrix<BaseFloat> *output) {
  int32 rows_out = NumFrames(wave.Dim(), opts_.frame_opts),
      cols_out = opts_.num_ceps;
  output->Resize(rows_out, cols_out);
  Vector<BaseFloat> window;  // windowed waveform.
  Vector<BaseFloat> mel_energies; // energies for mel bins.
  for (int32 r = 0; r < rows_out; r++) {  // r is frame index..
    ExtractWindow(wave, r, opts_.frame_opts,
                  feature_window_function_, &window);
    srfft_->Compute(window.Data(), true);  // split-radix FFT
    ComputePowerSpectrum(&window);
    SubVector<BaseFloat> power_spectrum(window, 0, window.Dim()/2 + 1);
    mel_banks_.Compute(power_spectrum, &mel_energies);
    mel_energies.ApplyLog();  // take the log.
    SubVector<BaseFloat> this_mfcc(output->Row(r));
    // this_mfcc = dct_matrix_ * mel_energies [which now have log]
    this_mfcc.AddMatVec(1.0, dct_matrix_, kNoTrans, mel_energies, 0.0);
  }
}
```

End of this lecture