Topic labeling of speech in low-resource languages

Camille Goudeseune
Mark Hasegawa-Johnson

Beckman Institute, UIUC

Low-resource languages

About a dozen languages have transcribed audio.

300 **low-resource** languages lack transcribed audio.

6,700 **zero-resource** languages lack even monolingual text (Wikipedia).

Can't recruit people to transcribe 1000 hours of speech to train an ASR.

- Too slow: might need the ASR tomorrow, not next year.
- Too expensive to do for many languages at once, instead of as needed.

Mismatched crowdsourcing (MC)

Instead of transcriptions from native speakers:

- Split recordings into 1 second clips.
  (Short clips will be transcribed better.)
- Crowdsource 10 English speakers to transcribe each clip,
  as nonsense syllables 'fa kablee grakoo,' 'foga lee ragu,' etc.
- Restitch each recording's set of transcriptions.
- Build a phone lattice for each recording.

Implemented with OpenFst.

The error rate beats that of wrong-language ASR, for languages from many families. (*Adapting* wrong-language ASR using MC can reduce error rate, but that's another talk.)

References

Mismatched Crowdsourcing and Mismatched Transcription

- Open-source software
  `github.com/uiuc-sst/PTgen`
- "Acquiring speech transcriptions using MC"
  `dl.acm.org/citation.cfm?id=2887007.2887182`
- "Adapting ASR for under-resourced languages using MT"
  `doi.org/10.1109/ICASSP.2016.7472797`

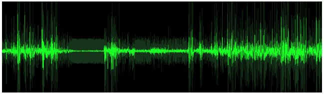
Goal and motivation

Given dozens of hours of recorded speech, quickly make sense of it by tagging it with HADR **Topics**:

- Civil unrest / widespread crime
- Elections / politics
- Evacuation
- Food supply
- Infrastructure
- Medical assistance
- Shelter
- Terrorism / extreme violence
- Urgent rescue
- Utilities / energy / sanitation
- Water supply

Task

Input:
recorded speech
(radio, Internet radio,
videos from social media)

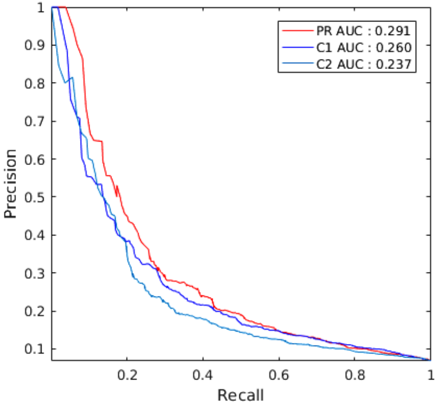


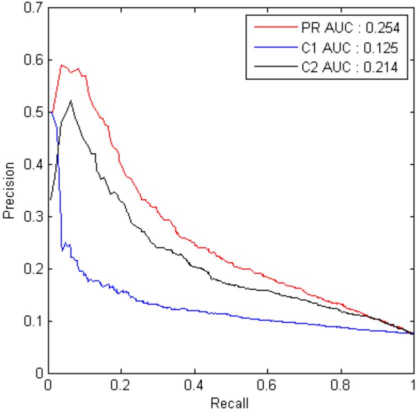
Output:
to each brief segment,
assign a probability $0 \leq P \leq 1$
for each topic.

$(0.14, 0.003, 0.01, \ldots, 0.31)$

Performance evaluation

Task performance is measured with a precision-recall **curve**, so topic labelers can trade off precision vs. recall.





Performance evaluation: Area Under Curve

- Finely sweep a threshold t from 0 to 1. At each step:
  - For each utterance,
    - Define the guess as those topics whose probability exceeds t.
    - Count the guess's true positives, false positives, and false negatives.
    - Accumulate those into running totals.

  Yes, utterances get unequal weights! That's OK.
  - From those running totals, calculate the precision and recall (one point on the curve).
- Measure the AUC: sum the area of lower trapezoids.

Repurposing MC for topic detection

For each utterance, PTgen outputs an ASR-like phone lattice.
To transform that into a vector of topic probabilities:

- From a low-resource word list and G2P, make a pronunciation lexicon, a map from words to phone sequences.
- Compose the phone lattice with that lexicon to make a word lattice.
- Traverse best paths through that word lattice to make word sequences, aka probable transcriptions.
- Send the transcriptions to a topic evaluator.

To label topics for a huge phone lattice:

- Modify all four of these steps. . .

Common steps that end each approach

- Get a few hundred probable transcriptions of the utterance.
- Machine-translate each one to English.
- Send each English transcription to a topic evaluator ("Woeful! My crops all burned down are." $\mapsto$ "Food Supply")
- Combine those evaluations to get each topic's net probability of being applicable to this utterance.

Next, five approaches. . .

1. Limit the vocabulary.

In agglutinative languages, the lexicon can exceed 1M words, too big to compose with a phone lattice.

Failed workaround: **keyword spotting.** Sequentially composing single- or thousand-word lexicons still eventually exhausts RAM.

Simpler workaround: **restrict the lexicon** to only a few thousand words, words that are statistically more relevant to the Topics.
Although this increases overall WER, many of these word errors don't hurt topic labeling.

2. `fstviterbisearch`, not `fstcompose`.

To bound the size of the composed word lattice
and bound how much RAM the composing requires,

instead of pruning *after* composition,
run Viterbi beam search *during* composition.

Customize OpenFST's `compose.h`:
when adding an outgoing arc from a state,
maintain a `std::priority_queue` to add only the n least costly arcs.
This keeps each state's outdegree $\leq n$.

3. "Joker" phone.

To shrink a phone lattice:
At each state, keep only the 5 most likely outgoing arcs.
Collapse the rest into one joker-labeled arc, of the same net likelihood, that matches *any* phone.

Why not just prune all the low-likelihood arcs?

Pruning bogusly removes the noise inherent in probabilistic transcription.
Instead of *removing* that noise, the joker arc *replaces* it with uniformly distributed noise.

4. `fstrandgen` *before* `fstcompose`.

Conventional:
Compose the phone lattice with the lexicon to get a word lattice.
Then take n random paths through the word lattice
to get n word sequences.

Cheat:
First take n random paths through the phone lattice to make a
$500\times$ **tinier** phone lattice: initial state with arcs to n chains.
Then compose that tiny lattice with the lexicon
to get n word sequences.

Information is lost: the composed FST's sausages are fewer and smaller.
But the cheat works, and it's $10\times$ faster.

5. Bypass translation to English.

- Calculate a topic's probability by naively counting words found in a few hundred topic-related phrases *in the original language,* then normalizing all topic probabilities to sum to unity.
- For any word that appears in phrases for multiple topics, amortize its weight over all those topics.
- Low quality, but blazingly fast because no translation is needed.

Interim results

Work in progress. No hard numbers yet.
Languages with ground-truth scoring data are Uyghur and Uzbek.

- `fstviterbisearch` quickly composes FSTs as big as 10 GB, using well under 32 GB of RAM.
- Aggressively constraining the lexicon to as few as 500 words scores reasonably well, though not as high as the state of the art.
- `fstrandgen` before `fstcompose` also scores surprisingly well.
- Naive topic spotting in the original language scores poorly, but runs quickly.

Acknowledgements

This research is supported by DARPA's program LORELEI,
Low Resource Languages for Emergent Incidents.