Linguistic Analysis

From lists of words to how to say them:
- segments, duration, F0.

☐ Lexical look up

☐ Prosody generation:
  - phrasing
  - intonation: accents and F0 contours
  - durations
  - power
Part of speech tagging

☐ Nouns, verbs, etc

☐ Needed for lexical lookup

☐ Needed for phrase prediction

☐ Most likely POS tags for a word gives:
  – 92% correct (+/-)

☐ Content/function word distinction easy
  – (and maybe sufficient)
Use standard Ngram model

find $T_1, \ldots, T_n$ that maximize $P(T_1, \ldots, T_n \mid W_1, \ldots, W_n)$

$\approx \prod_{k=1}^{n} \frac{P(T_k \mid T_{k-1}, \ldots, T_{k-N+1})P(W_k \mid T_k)}{P(W_k)}$

- Lexical Probabilities
  - For each $W_k$ hold converse probability $P(W_k \mid T_k)$.

- Ngram
  - $P(T_k \mid T_{k-1}, \ldots, T_{k-N+1})$

- Viterbi decoder to find best tagging
Building a tagger

- From existing tagged corpus:
  - find $P(T \mid W)$ by counting occurrences
  - Build trigram from data

- But if no existing tagged corpus exists:
  - tag one by hand, or ...
  - tag it with naive method
  - collect stats for probabilistic tagger
  - re-label and re-collect stats
  - repeat until done
What tag set?

But in synthesis we only need n,v,adj

Reduce $\rightarrow$ build models $\rightarrow$ predict
build models $\rightarrow$ predict $\rightarrow$ reduce

<table>
<thead>
<tr>
<th>Tagset</th>
<th>POS Ngram model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uni</td>
</tr>
<tr>
<td>ts45</td>
<td>90.59%</td>
</tr>
<tr>
<td>ts22</td>
<td>95.22%</td>
</tr>
<tr>
<td>45/22</td>
<td></td>
</tr>
</tbody>
</table>
Lexicon

- Pronunciation from words plus POS tag
- In Festival includes stress and syllabification:
  - ("project" n (((p r aa jh) 1) ((eh k t) 0)))
  - ("project" v (((p r ax jh) 0) ((eh k t) 1)))
- But need extra flags for (some homographs)
Lexicon

☐ Lexicon *must* give pronunciation:
  - what about morphology

☐ Festival lexicons have three parts:
  - a large list of words
  - a (short) addenda of words
  - letter to sound rules for everything else
Different languages

- (US) English:
  - 100,000 words (CMUDICT)
  - 50 words in addenda (modes modify this)
  - Statistically trained LTS models

- Spanish:
  - 0 words in large list
  - 50 words (symbols) in addenda
  - Hand written LTS rules
Letter to Sound rules

If language is “easy” do it by hand

☐ ordered set of rules
( LEFTCONTEXT [ ITEMS ] RIGHTCONTEXT = NEWITEMS )

☐ For example:
( _edge_ [ c h ] C = k )
( _edge_ [ c h ] = ch )

☐ Often rules are done in multiple-passes:
  - case normalization
  - letter to phones
  - syllabification
Letter to Sound rules

If language is “hard” train them

☐ For English rules by hand can be done but
  – its is a skilled job
  – time consuming
  – rule interactions are a pain

☐ Need it for new languages/dialects NOW
Letter to phone alignment

What is the alignment for

checked - ch eh k t

one-to-one letter/phone pairs desirable

<table>
<thead>
<tr>
<th>c</th>
<th>h</th>
<th>e</th>
<th>c</th>
<th>k</th>
<th>e</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>_</td>
<td>eh</td>
<td>_</td>
<td>k</td>
<td>_</td>
<td>t</td>
</tr>
</tbody>
</table>

Need to find best alignment automatically
Letter to phone alignment algorithms

Epsilon scattering algorithm (expectation maximization)

- find all possible alignments
- estimate prob(L,P) on each alignment
- iterate

Hand seeded approach

- Identify all valid letter/phone pairs e.g.
  - c → _ k ch s sh
  - w → _ w v f

- find all alignments (within constraints)
- find score of L/P
- find alignment with best score

SMT type alignment

- Use standard IBM model 1 alignment
- Works “reasonably” well
Alignments – comments

☐ Sometimes letters go to more than one phone, e.g.
   - x → k-s, cf. “box”
   - l → ax-l, cf. “able”
   - e → y-uw, cf. “askew”
   dual-phones added as phones

☐ Some alignments aren’t sensible
   - dept → d ih p a a r t m ah n t
   - lieutenant → l eh f t eh n ax n t
   - CMU → s iy eh m y uw
   But less than 1%
Alignment comparison

Models (described next) on OALD held-out test data

<table>
<thead>
<tr>
<th>Method</th>
<th>Letters</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epsilon scattering</td>
<td>90.69%</td>
<td>63.97%</td>
</tr>
<tr>
<td>Hand-seeded</td>
<td>93.97%</td>
<td>78.13%</td>
</tr>
</tbody>
</table>

Hand-seeded takes time, and a little skill so fully automatic would be better.
Training models

- We use decision trees (CART/C4)
- Predict phone (dual or epsilon)
- Window of 3 letters before, 3 after
  
  ```
  # # # c h e c  →  c h
  c h e c k e d  →  _
  ```
## Results

On held out test (every 10th word)

<table>
<thead>
<tr>
<th>Lexicon</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Letters</td>
</tr>
<tr>
<td>OALD</td>
<td>95.80%</td>
</tr>
<tr>
<td>CMUDICT</td>
<td>91.99%</td>
</tr>
<tr>
<td>BRULEX</td>
<td>99.00%</td>
</tr>
<tr>
<td>DE-CELEX</td>
<td>98.79%</td>
</tr>
<tr>
<td>Thai</td>
<td>95.60%</td>
</tr>
</tbody>
</table>

Reflects language and lexicon coverage.
# Results (2)

<table>
<thead>
<tr>
<th>Stop</th>
<th>Correct Letters</th>
<th>Correct Words</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>92.89%</td>
<td>59.63%</td>
<td>9884</td>
</tr>
<tr>
<td>6</td>
<td>93.41%</td>
<td>61.65%</td>
<td>12782</td>
</tr>
<tr>
<td>5</td>
<td>93.70%</td>
<td>63.15%</td>
<td>14968</td>
</tr>
<tr>
<td>4</td>
<td>94.06%</td>
<td>65.17%</td>
<td>17948</td>
</tr>
<tr>
<td>3</td>
<td>94.36%</td>
<td>67.19%</td>
<td>22912</td>
</tr>
<tr>
<td>2</td>
<td>94.86%</td>
<td>69.36%</td>
<td>30368</td>
</tr>
<tr>
<td>1</td>
<td>95.80%</td>
<td>74.56%</td>
<td>39500</td>
</tr>
</tbody>
</table>
An example tree

For letter V:
if (n.name is v)
    return _
    if (n.name is #)
        if (p.p.name is t)
            return f
        return v
    return v
if (n.name is s)
    if (p.p.p.name is n)
        return f
    return v
return v
Stress assignment

The phone string isn’t enough
– train separate stress assignment
– make stressed/unstressed phones (eh/eh1)

<table>
<thead>
<tr>
<th></th>
<th>LTP+S</th>
<th>LTPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>L no S</td>
<td>96.36%</td>
<td>96.27%</td>
</tr>
<tr>
<td>Letter</td>
<td>—</td>
<td>95.80%</td>
</tr>
<tr>
<td>W no S</td>
<td>76.92%</td>
<td>74.69%</td>
</tr>
<tr>
<td>Word</td>
<td>63.68%</td>
<td>74.56%</td>
</tr>
</tbody>
</table>

– includes POS in LTPS (71.28% word, without)
– still missing morphological information though
Does it really work

Analysis *real* unknown words

In 39923 words in WSJ (Penn Treebank),
1775 (4.6%) not in OALD

<table>
<thead>
<tr>
<th></th>
<th>Occurs</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>names</td>
<td>1360</td>
<td>76.6</td>
</tr>
<tr>
<td>unknown</td>
<td>351</td>
<td>19.8</td>
</tr>
<tr>
<td>American spelling</td>
<td>57</td>
<td>3.2</td>
</tr>
<tr>
<td>typos</td>
<td>7</td>
<td>0.4</td>
</tr>
</tbody>
</table>
“Real” unknown words

Synthesize them with LTS models and *listen*.

<table>
<thead>
<tr>
<th>Stop</th>
<th>Lexicon Test set</th>
<th>Unknown Test set</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74.56%</td>
<td>62.14%</td>
<td>39500</td>
</tr>
<tr>
<td>4</td>
<td>65.17%</td>
<td>67.66%</td>
<td>17948</td>
</tr>
<tr>
<td>5</td>
<td>63.15%</td>
<td>70.65%</td>
<td>14968</td>
</tr>
<tr>
<td>6</td>
<td>61.65%</td>
<td>67.49%</td>
<td>12782</td>
</tr>
</tbody>
</table>

Best lex test is *not* best for unknown
Bootstrapping Lexicons

- Lexicon is largest (size/ expensive) part of system
- If you don’t have one:
  - use someone else’s
- Building your own takes time
Bootstrapping Lexicons

- Find 250 most frequent words:
  - build lexical entries for them
  - ensure letter coverage in base set
  - Build lts rules from this base set

- Select articles of text

- Synthesis each unknown word
  - listen to the synthesized version
  - add correct words to base list
  - correct incorrect words and add to base list
  - rebuild lts rules with larger list
  - repeat
Bootstrapping Lexicons: tests

- Using CMUdict as “oracle”
  - start with 250 common words
  - 70% accuracy
  - 25 iterations gives 97% accuracy (24,000 entries)

- Using DE-CELEX:
  - base 350 words: 35% accurate
  - ten iterations ot 90% accurate

- Real “new” lexicons:
  - Nepali
  - Celex (English) 12,000 entries at 98%
Dialect Lexicons

- Need new lexicons for each dialect:
  - expensive and difficult to maintain

So build dialect independent lexicon

- Build lexicon with “key vowels”:
  - the vowel in coffee

- vowels in pull and pool:
  - In Scots English map to same
  - In Southern (UK) English map to different

- word-final ‘r’
  - delete in Southern UK English

- Plus specific pronunciation differences:
  - leisure, route, tortoise, poem
Post-lexical rules

- Some pronunciations require context
- For example “the”
  - before vowel dh iy
  - before consonant dh ax
- Taps in US English
- Nasals in Japanese (“san” to “sam”)
- Liaison in French
- Speaker/style specific rules:
  - vowel reduction
  - contractions
  - and others
Exercises for April 10th

3 is optional

1. Add a post-lexical rule to modify the pronunciation of “the” before vowels, can you make it work for UK and US English.

2. Use SABLE markup to tell a joke.

3. Write letter to sound rules to pronounce Chinese proper names (in romanized form) in (US) English.
Variable `poslex_rules_hooks` is list of functions run on utterance after lexical lookup

```
(define (postlex_thethee utt)
  (mapcar
    (lambda (seg)
      (if word is the, this is last segment, and next segment is a vowel
          change vowel in segment)
    )
    (utt.relation.items utt 'Segment)))

(set! postlex_rules_hooks (cons postlex_thethee postlex_rules_hooks))
```

Features are:

- `R:SylStructure.parent.parent.name`
- `R:SylStructure.n.name`
- `n.name`

Test is with

```
(set! utt1 (SayText "The oval table."))
(set! utt2 (SayText "The round table."))
(utt.features utt1 'Segment '(name))
```
Telling a joke

They say telling a joke is in the timing.

☐ Use different speakers, breaks, etc to get the joke over.

☐ A sample joke is in

   http://www.cs.cmu.edu/~awb/11752/joke.txt

☐ A useful audio clip is in

   http://www.cs.cmu.edu/~awb/11752/laughter.au