Cache adaptation with a part-of-speech model for word prediction

Keith Trnka
SIG-AI 2010-4-5

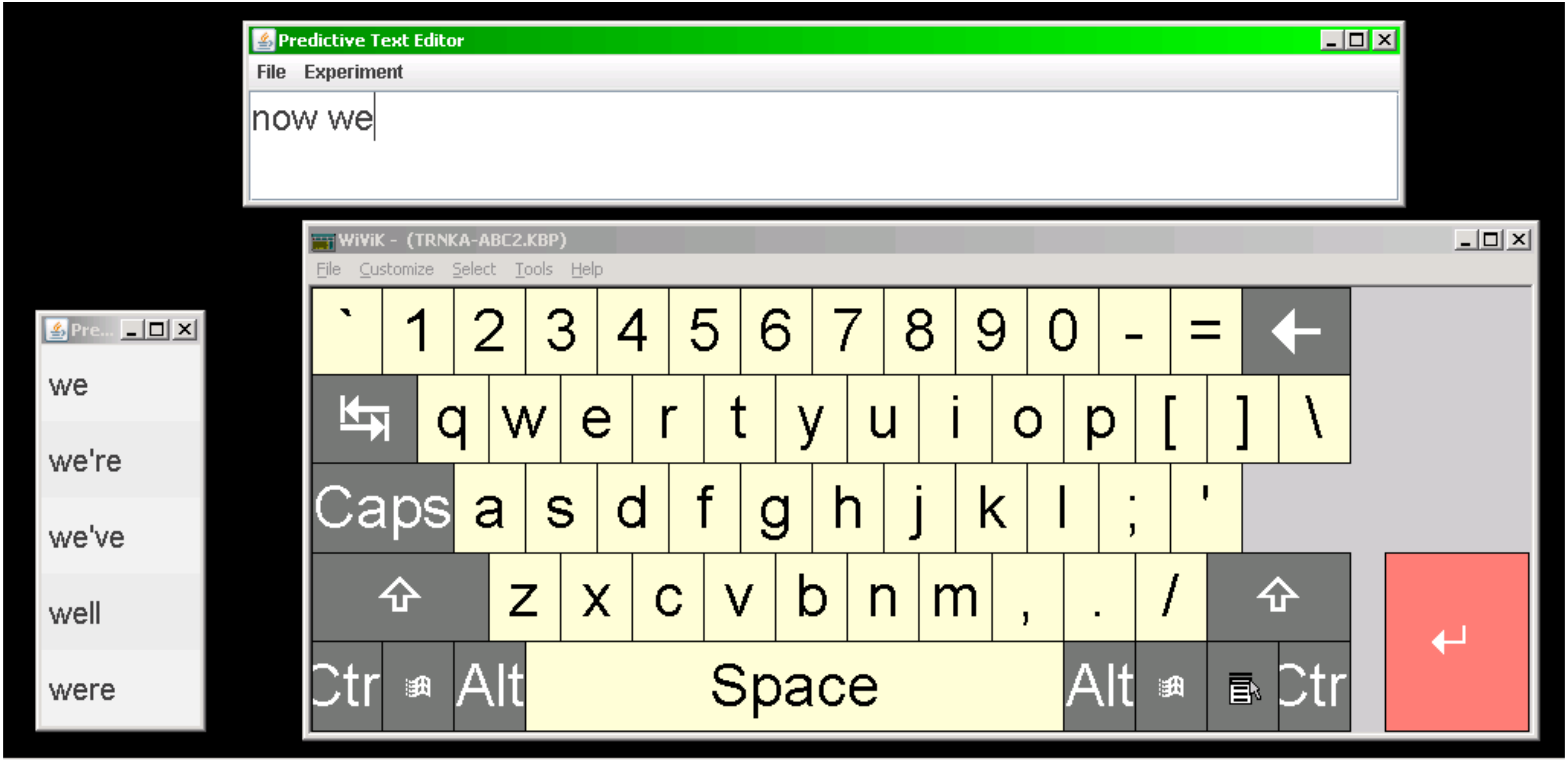
keywords

- language modeling
  - adaptive language modeling
  - cache
  - part of speech modeling/tagging
- augmentative and alternative communication
  - word prediction

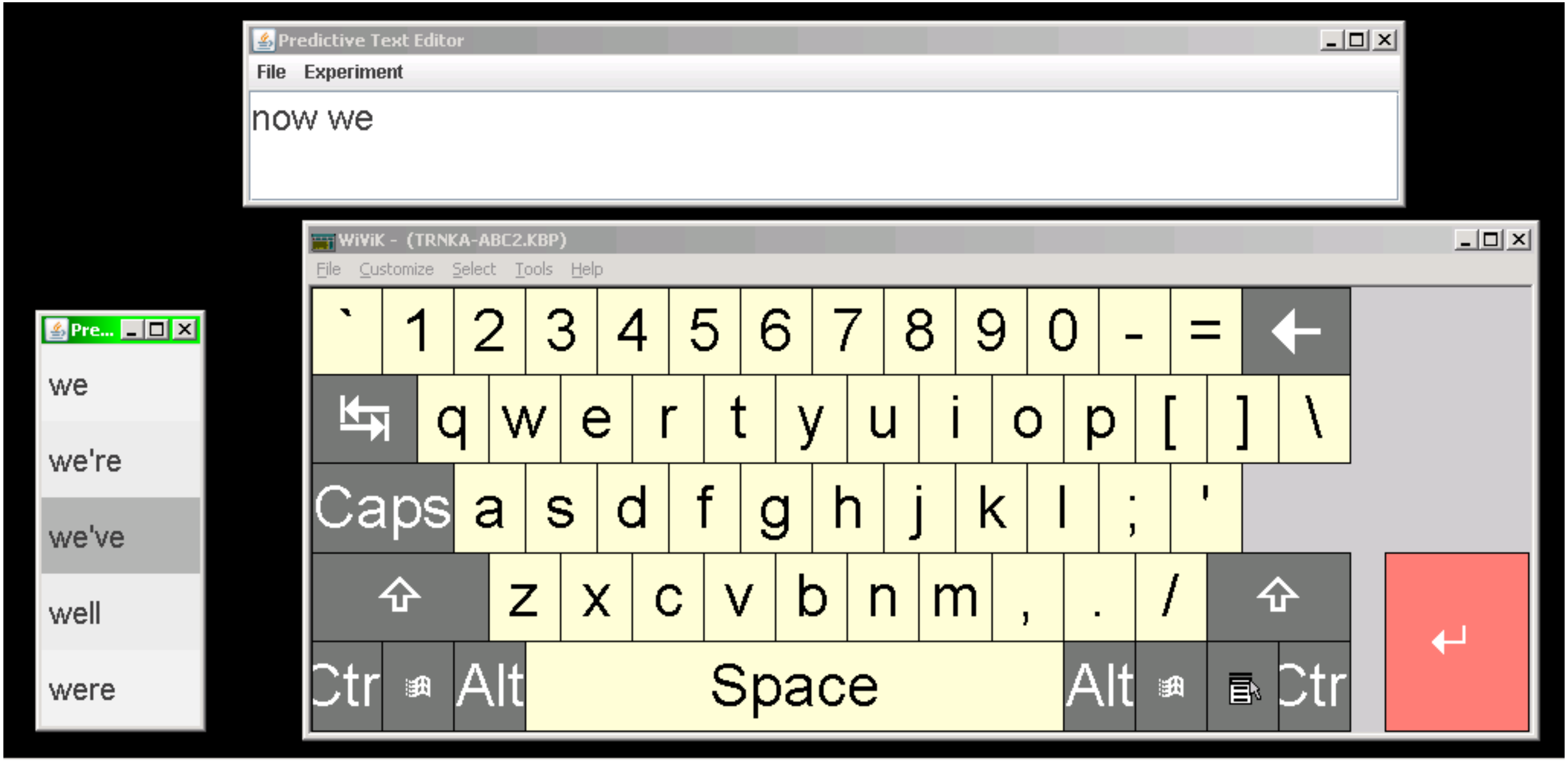
AAC Background

- augmentative and alternative communication (AAC)
  - speech + motor impairment
  - ***AAC device*** ~= tablet PC + custom "typing" interface + word prediction + speech synthesis
  - main problem: communication rate

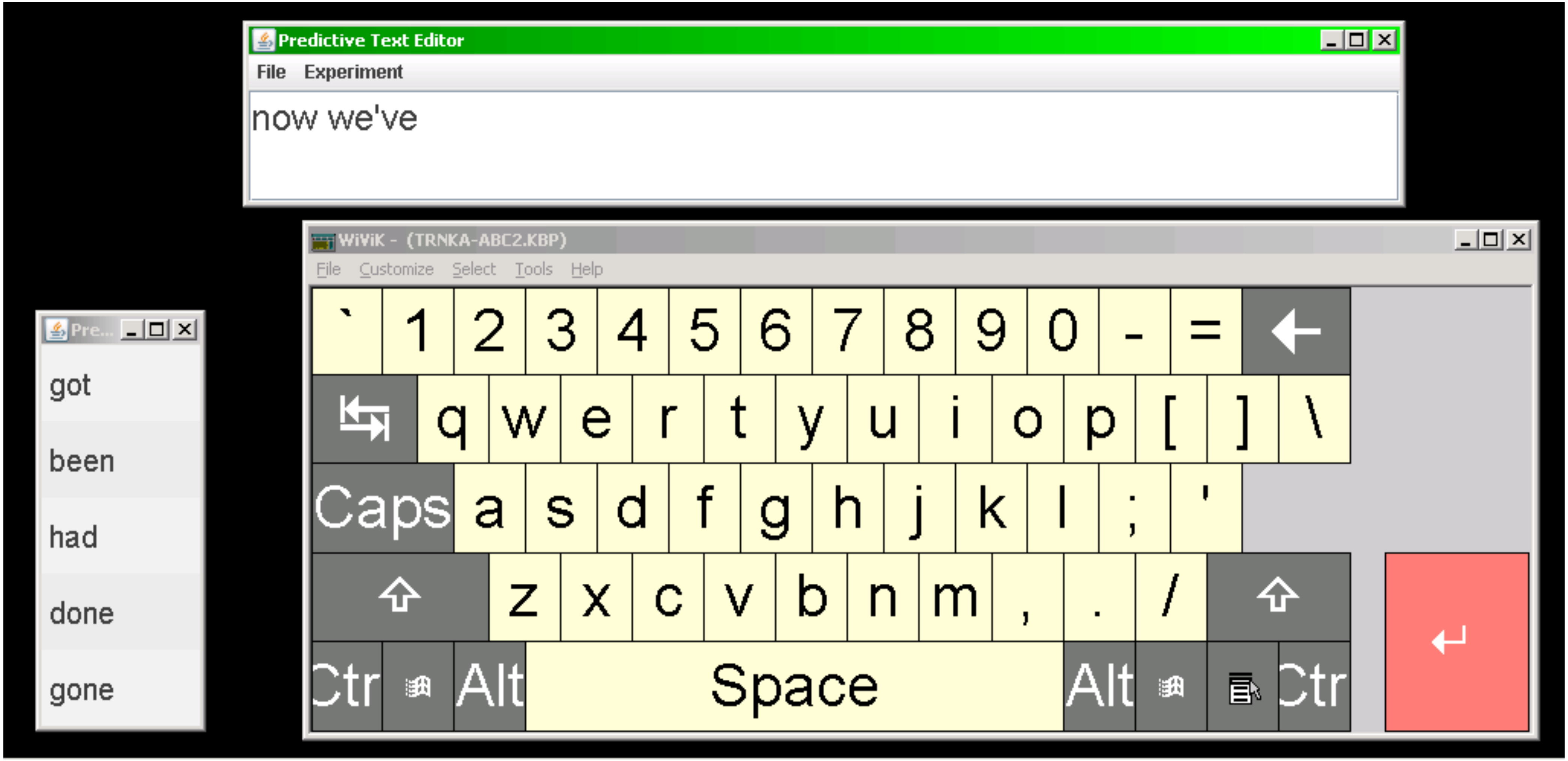
Example


Predictive Text Editor
File Experiment
now we
WiViK - (TRNKA-ABC2.KBP)
File Customize Select Tools Help
we
we're
we've
well
were
Caps
Space
Alt
Ctr

Example


Predictive Text Editor
File Experiment
now we
WiViK - (TRNKA-ABC2.KBP)
File Customize Select Tools Help
` 1 2 3 4 5 6 7 8 9 0 - =
q w e r t y u i o p [ ] \
Caps a s d f g h j k l ; '
z x c v b n m , . /
Ctr Alt Space Alt Ctr
Pre...
we
we're
we've
well
were

Example


Predictive Text Editor
File Experiment
now we've
WiViK - (TRNKA-ABC2.KBP)
File Customize Select Tools Help
` 1 2 3 4 5 6 7 8 9 0 - =
q w e r t y u i o p [ ] \
Caps a s d f g h j k l ; '
z x c v b n m , . /
Ctr Alt Space Alt Ctr
Pre...
got
been
had
done
gone

Language modeling background

- language model = mapping of words *w* to probabilities given some context *h*
- ngram models
  - unigram = a single probability table, ignores context

$$P(w) = \frac{freq(w)}{\sum_i freq(w_i)}$$

Language modeling background

- bigram = separate probability table for each possible previous word w_{-1}

$$P(w \mid w_{-1}) = \frac{freq(w_{-1}, w)}{freq(w_{-1})}$$

Language modeling background

- trigram = separate probability table for each pair of possible previous words w_{-1}, w_{-2}

$$P(w \mid w_{-1}, w_{-2}) = \frac{freq(w_{-2}, w_{-1}, w)}{freq(w_{-2}, w_{-1})}$$

Language modeling background

- an so on for higher order ngrams...
- smoothing/backoff/etc
    - how do we deal with $P(w \mid w_{-1})$ when
        - w wasn't seen in training
        - w was in training but w_{-1} wasn't
    - need (meaningful) probabilities for these

Language modeling problems

- very data-sensitive
  - training/testing similar = good performance
  - training/testing dissimilar = bad performance
  - not much training data = bad performance
- this talk = small email corpus
  (28k words, 117 documents)

LMs in word prediction

- rank the words for the given context
- filter out anything that doesn't match the prefix (if any)
- take the top few
- reality: to get work done, you need to avoid sorting the whole vocabulary all the time

Evaluating word prediction

- user evaluation: communication rate
  - tedious to evaluate
- automated: ***keystroke savings***
  - closest automatic metric to users
- automated: perplexity
  - common in speech recognition, fast

Evaluating word prediction

$$KS = \frac{chars - keystrokes}{chars} \times 100\%$$

- keystroke savings
  - assume multi-keystroke characters are very rare in AAC
  - assume that the simulated user is perfect
  - common values: 45-60%
  - theoretical max ~80% for most data

Evaluating word prediction

$$KS = \frac{chars - keystrokes}{chars} \times 100\%$$

- we assume 5 predictions (more = higher KS)

Cache modeling

- (iteratively) learn some ngram model on testing data
  - caveat: in this work, I’m only interested in using part of a testing document on the rest of that document
  - not interested in making a big language model containing ALL testing data

Cache modeling

- typical plain cache = unigram model, iteratively trained
  - combining the cache model with the baseline model is most of the work

Cache modeling

- What's the linguistic intuition?
  - caches are traditionally results-motivated rather than linguistically
  - two linguistic aspects of caches
    - recency promotion
    - vocabulary learning

Evaluating cache models

- Split evaluation
  - record keystroke savings on seen and unseen words separately
- Keep in mind
  - cache models are more beneficial the less similar training/testing is

Word unigram cache

- simple combination methods:
  - place cache predictions after normal predictions (if there's any space)
  - place cache predictions before
- caveat: perplexity not appropriate for simple methods

Word unigram cache

Run	Overall	Seen	Unseen
Baseline	48.217%	54.934%	0

Word unigram cache

Run	**Overall**	**Seen**	**Unseen**
Baseline	48.217%	54.934%	0
unigram cache (after)	48.718% (+0.501)	54.941%	4.038%

Word unigram cache

Run	Overall	Seen	Unseen
Baseline	48.217%	54.934%	0
unigram cache (after)	48.718% (+0.501)	54.941%	4.038%
unigram cache (before)	46.359% (-1.858)	51.880%	6.724%

Word bigram cache

- bigrams can be used in simple manners too
  - difference: more sparse, more accurate

Word bigram cache

Run	Overall	Seen	Unseen
Baseline	48.217%	54.934%	0
bigram cache (before)	49.099% (+0.882)	55.525%	2.963%

Word bigram cache

Run	**Overall**	**Seen**	**Unseen**
Baseline	48.217%	54.934%	0
bigram cache (before)	49.099% (+0.882)	55.525%	2.963%
bigram cache (after)	48.399% (+0.182)	54.941%	1.431%

Combining models

- combining the cache model can be more advanced too
  - linear combination

$$P(w \mid h) = 0.5 * P_{static}(w \mid w_{-1}) + 0.5 * P_{cache}(w)$$

  - geometric combination

$$P(w \mid h) = P_{static}(w \mid w_{-1})^{0.5} * P_{cache}(w)^{0.5}$$

Existing cache models

- Jelinek et al. (1991) *A dynamic language model for speech recognition*
  - cache model using linear combination of:
    - static trigram, bigram, unigram
    - cache trigram, bigram, unigram
  - improved results substantially

Existing cache models

- Kuhn and De Mori (1990) *A Cache-Based Natural Language Model for Speech Recognition*
  - starting point is part of speech ngrams rather than word ngrams
    - aka the guts of a Markov model POS tagger
  - bigger improvement, so we'll apply this to word prediction

POS model

- requires a corpus that's labeled with part of speech information
  - really we just used an existing tagger to tag it and added some cleanup rules

POS model

- assuming we know the tag of the previous word is t_{-1} and are using a bigram POS model

$$P(w \mid tag(w_{-1}) = t_{-1}) = \sum_{t \in tagset} P(tag(w) = t \mid t_{-1}) * P(w \mid t)$$

- reality: we don't know the previous tag, but
  - we have guesses and probabilities for each guess as a part of the model

POS model

$$P(w \mid w_{-1}) = \sum_{t_{-1} \in tagset} P(t_1, t_2, \ldots, t_{-1}) * \sum_{t \in tagset} P(t \mid t_{-1}) * P(w \mid t)$$

- markov model tagging iteratively tracks the best tagging t_1 t_2 ... for each ending tag t_{-1}
- we will refer to this as a POS ngram model, where the ngram order is the order of the transition probability P(t | ...)

POS model - training

- Where does $P(t \mid t_{-1})$ come from?
  - measured on the training data
    - $f(t_{-1}, t) / f(t_{-1})$ and variations thereof
- Where does $P(w \mid t)$ come from?
  - measured on training data
    - $f(tag(w) = t) / f(t)$ and variations thereof

Why do we care?

$$P(w \mid t_{-1}) = \sum_{t \in tagset} P(t \mid t_{-1}) * [\lambda * P_{static}(w \mid t) + (1 - \lambda) * P_{cache}(w \mid t)]$$

- Cache can record P(w | t)
  - combine this cache emission model with the baseline one
- Use the transition model $P(t \mid t_{-1}, \ldots)$ from the baseline model

Why do we care?

$$P(w \mid t_{-1}) = \sum_{t \in tagset} P(t \mid t_{-1}) * [\lambda * P_{static}(w \mid t) + (1-\lambda) * P_{cache}(w \mid t)]$$

- \lambda can be some arbitrary weight or an adaptive weight
    - we'll come back to this later

Why do we care?

$$P(w \mid t_{-1}) = \sum_{t \in tagset} P(t \mid t_{-1}) * [\lambda * P_{static}(w \mid t) + (1 - \lambda) * P_{cache}(w \mid t)]$$

- POS bigram model: $P_{cache}(w \mid t)$
  - number of possible conditions ~= 40-80
- word bigram model: $P_{cache}(w \mid w_{-1})$
  - number of possible conditions ~= 20,000? or many more

Why do we care?

$$P(w \mid t_{-1}) = \sum_{t \in tagset} P(t \mid t_{-1}) * [\lambda * P_{static}(w \mid t) + (1 - \lambda) * P_{cache}(w \mid t)]$$

- compared to unigram word cache:
  - cached words are predicted in grammatically appropriate situations
- compared to bigram word cache:
  - example: “the 802.11n” will enable “802.11n” after “an”

Development issues

- half of cache modeling - learning new words
  - but... the cache model requires POS tag... of unknown words
  - leveraging morphology in tagging becomes more important
- we update the cache after every sentence (cause then the sentence has a clear tagging)

POS cache - weights

- How to determine weights between the static and cache emission models?
  - static weights
    - e.g., arbitrary weights, tuned to held-out data
  - dynamic weights
    - tuned to the current document

POS cache - weights

- First experiment: arbitrary static weights vs dynamic weights
- Dynamic weight tuning - "win counting"
  - the static or cache model is given a "win" if it assigns higher probability to a word
  - normalize wins to get weights

POS cache - weights

Run	Overall	Seen	Unseen
Baseline	48.217%	54.934%	0
POS cache, 50/50 weights	49.368% (+1.151)	55.311%	6.707%
POS cache, dynamic	49.545% (+1.328)	55.551%	6.430%

POS cache - weights

- the devil's in the details...
  - initializing the number of wins per method has an impact
  - previous slide = 10n/10c

POS cache - weights

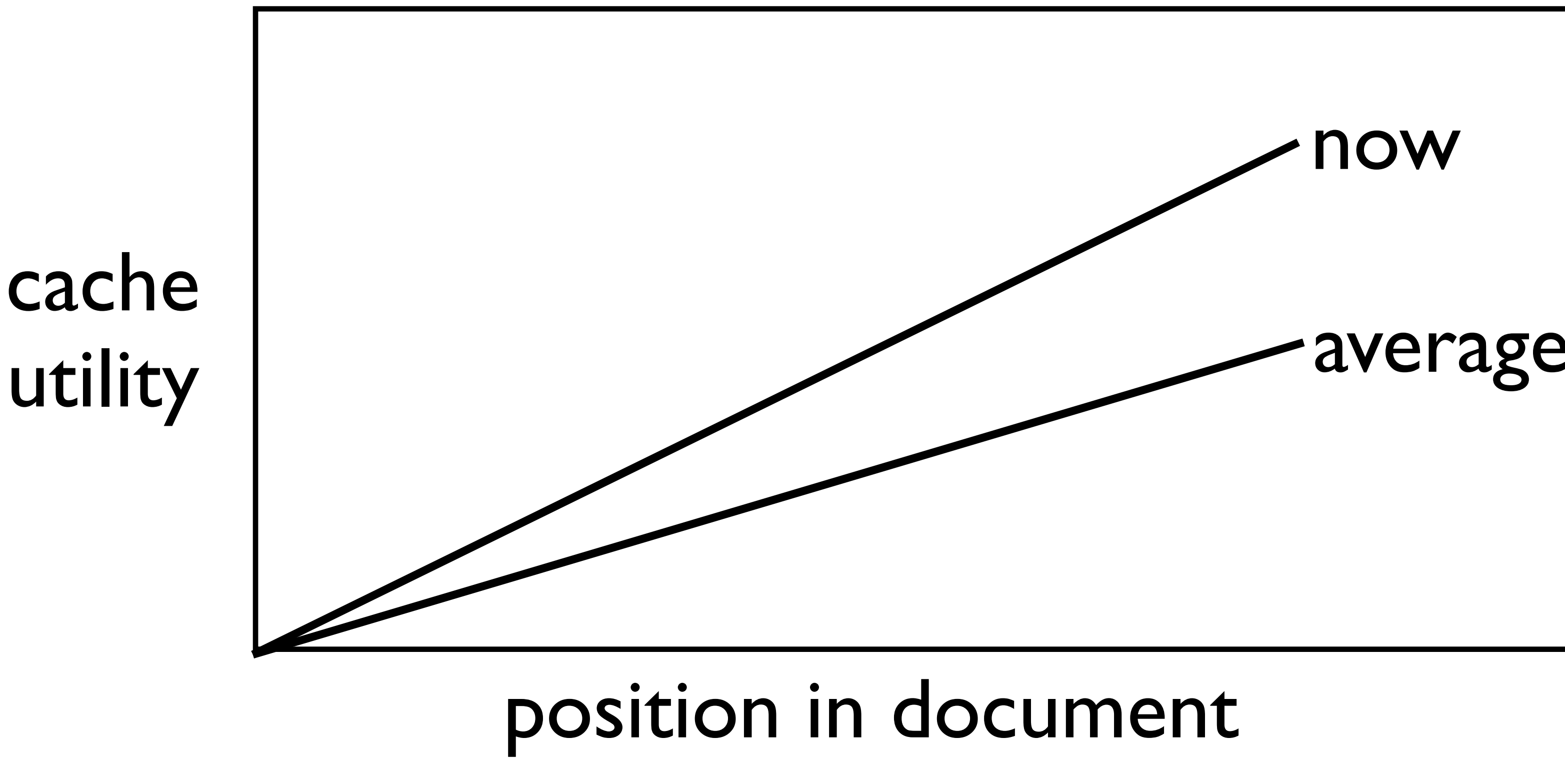
Run	Overall	Seen	Unseen
init = 10n/10c	49.545%	55.551%	6.430%
init = 1n/0c	49.519%	55.545%	6.260%
init = 20n/20c	49.531%	55.528%	6.481%

POS cache - weights

- sometimes your first guess is the best
- are dynamic weights really better than tuned static weights?
  - approximation: don't reset wins between documents
  - result: decreased keystroke savings

POS cache - weights

- Problem with dynamic weights:
  utility (right now) vs utility (average so far)



POS cache - weights

- Possible solutions
  - only count most recent K wins
  - exponential decay - just multiply all wins by 0.95 before counting a new win
    - any number in (0, 1) will work

Decayed weights

Run	Overall	Seen	Unseen
no decay	49.545%	55.551%	6.430%
0.95 decay	49.556%	55.553%	6.503%
0.90 decay	49.523%	55.519%	6.481%

POS-specific weights

- Kuhn and de Mori found a small benefit in POS-specific weights
  - cache is more useful for:
    - PRP: he, she, you, I, etc
    - NNP: names

POS-specific weights

- best with POS-specific (no decay): 49.515%
  - ... still lower than non-POS-specific weights
  - ... even with significant work

POS-specific weights

- example weights:
  - MD +0.136 (e.g., can, would, could)
  - IN +0.12 (e.g., of, in, for)
  - TO -0.11 (i.e., to)
  - VBZ -0.096 (e.g., contains, uses, shows)

New word derivation

- Given a word, could we automatically come up with derivations/inflections of the same base form?
- If we could, we could derive new words from the words we see in training
- Using the cache: re-distribute each word occurrence's frequency to itself and alternate derivations/inflections

Deriving new forms

- Per document:
  - model frequency as f(base, suffix)
  - build suffix co-occurrence frequencies by iterating over base forms
- Afterwards, given the input (base, $suffix_1$), compute the probability of (base, $suffix_2$) with this data

Modifying the cache

- given that we can compute $P(\text{suffix}_2 \text{ is valid} \mid \text{base}, \text{suffix}_1)$
- when a word occurs, iterate over all possible forms of the word
  - the frequency assigned to each form in the cache is the probability
  - exception: left the original form at 1 frequency for safety

Modifying the cache

- for each (possible) word form, consider all parts of speech using the suffix tagger
- set a threshold on the suffix and part of speech probabilities to keep the computation controlled

Decayed weights

Run	Overall	Seen	Unseen
basic cache (no decay)	49.545%	55.551%	6.430%
+ derivation	49.615%	55.563%	6.916%

Derivation cache

- considering it thinks “familie” is a valid word, the results are good
- and others...
  before -> befored
  united -> uniteds
  someone -> someoned, someoning
- (many aren’t this bad though)

- historical data says I won’t make it to this slide
- if I get here thanks for listening!

Perplexity

Evaluating LMs

$$PP(w_1, \ldots, w_N) = \frac{1}{\sqrt[N]{\prod_i P(w_i \mid h)}}$$

- perplexity
  - faster to compute - doesn't require ranking
  - intuition:
    - 1 / (geometric mean probability per word)
    - the number of words we're confused amongst (on average)

Evaluating LMs

$$PP(w_1, \ldots, w_N) = 2^{-\frac{1}{N}\sum_i log_2 P(w_i|h)}$$

- perplexity
  - reality: have to compute in log-space
  - is affected by the probability of unknown words (which doesn't affect KS)

Evaluating LMs

- word prediction
  - keystroke savings preferred
  - perplexity sometimes useful