Effective Phrase Prediction

Arnab Nandi & H.V. Jagadish

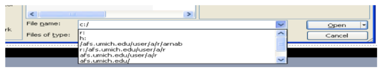
University of Michigan

{**arnab**, jag}@umich.edu

Autocompletion

- Popular UI feature to assist input
- For current input, *unobtrusively* present a set of options that complete the input
- Great when
  - cost of input is high
  - input is repetitive

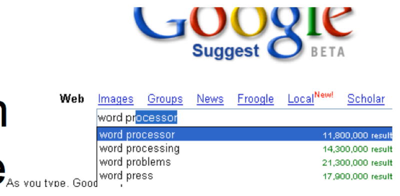
Applications



```
c:/
r:
h:
/afs.umich.edu/user/a/r/arnab
r:/afs.umich.edu/user/a/r
afs.umich.edu/user/a/r
afs.umich.edu/
```

OS

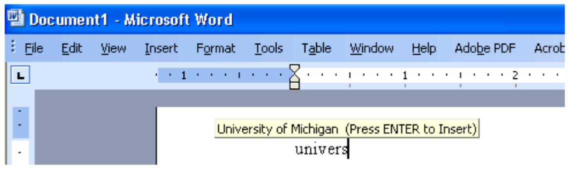
Search Engine



```
word processor
word processor          11,800,000 result
word processing         14,300,000 result
word problems           21,300,000 result
word press              17,900,000 result
```

As you type, Good

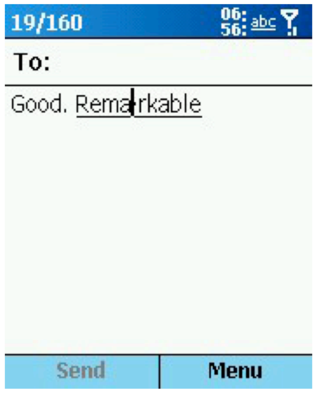
Word Processor



University of Michigan (Press ENTER to Insert)

univers

Mobile Phone



Good. Remarkable

- Autocomplete is everywhere : input reduction
- But, typical autocompletion is still at word-level. We can do better: Why not *phrase* level? Just use words as tokens!
- Words provide much more information to exploit for prediction (context, phrase structures)

An Example

- Email Composition
  - Most text is **predictable** and **repetitive**
    - “Thank you very much”, “Please let me know if you have any”
  - Prob(“Thank you very much” | “Thank”) ~= 1
  - Why not suggest the phrase after the first word?

- Even more relevant for
  - Customer Service *(have you tried turning it off and on again?)*
  - Programming *(System.out.println)*
  - Constrained input devices (mobile, accessible) *(T9 / SMS, Dasher)*

Challenges

- Number of phrases is large:

  n(vocabulary) >> n(alphabet)

  n(Phrases) = $O(\text{vocabulary}^{\text{phrase length}})$

- Length of phrase is unknown

- How to evaluate a suggestion mechanism?

Outline

- Motivation
- ***Data Model***
- Evaluation
- Experiments
- Extensions

Problem Definition

.. required, documents, are, attached, please, let, me, know, if, you, have.

Problem Definition

- Text input = stream of words

.. required, documents, are, attached, please, let, me, know, if, you, have.

Problem Definition

- Text input = stream of words

… w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, .

Problem Definition

- Text input = stream of words

… w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, .

Problem Definition

- Text input = stream of words

… w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, .

prefix ***p***

Words for which we return a set of completions *r*

Problem Definition

- Text input = stream of words

… w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, .

prefix ***p***

Words for which we return
a set of completions *r*

Problem Definition

- Text input = stream of words

… **w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14…**

prefix ***p***

Words for which we return a set of completions *r*

completion ***r*** ∈ ***R***

Set of word phrases, such that *prob(p,r)* is maximized for each *r.*

Problem Definition

- Text input = stream of words

… w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14…

prefix ***p***

Words for which we return a set of completions *r*

completion ***r*** $\in$ ***R***

Set of word phrases, such that *prob(p,r)* is maximized for each *r.*

- ***R = query(p)***
- Need data structure that can
  - store completions efficiently
  - support fast querying

An n-gram Data model

- ***`R = query(p)`*** : $r \in R$, prob(p,r) is maximized
- Hence, we need to store all *frequent* (p,r)

… **w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14…**

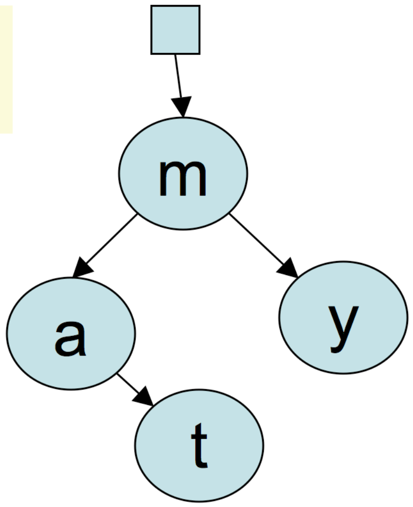
- How to train?
- Consider text as a Markov chain
- Consider a sliding window over this chain

- The size of the window = *N*
  - Upper limit for the size of a frequent phrase
  - Nth order Markov assumption, w_i is independent of w_{i+N+1}

An n-gram model

- Problems
  - how do we detect frequent phrases?
    - No start / end markers
    - Storing *all* frequent phrases is complicated
- Good data structure properties
  - Efficient storage of frequent phrases
  - Fast query times

FussyTree



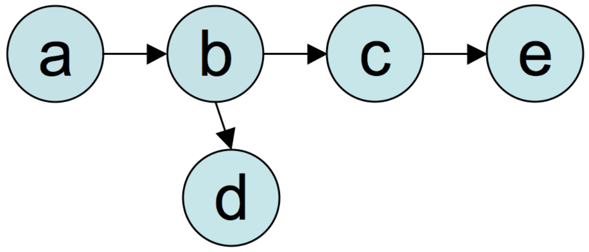
- Basic data structure to “completion” problems = trie
- Phrase trie : every node = word
- Problem: we can’t store ALL phrases
- Solution: be *fussy* about the phrases added

FussyTree Construction

- Naïve algorithm (Pruned Count Suffix Tree)
  - Construct a frequency based phrase trie
  - Prune all nodes with frequency < threshold τ
  - *Problems: ALL text in tree!!*
- FussyTree Construction
  - Filter out infrequent phrases even before adding to the tree.

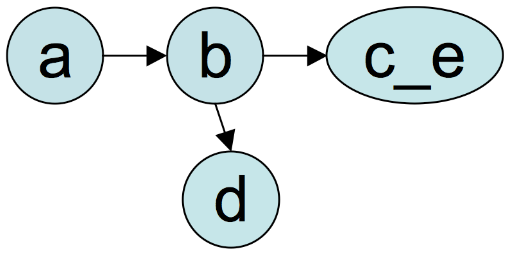
FussyTree Construction

- Three phase construction
- Phase 1: Generate Frequency counts
- Phase 2: Construct “Fussy” Suffix Tree
  - Phrases and their prefixes are considered for addition only if they are ***frequent***
    - Frequency check optimization: check all substrings of candidate phrase for frequency
  - All paths in tree are thus frequent phrases
- Phase 3: Telescope paths



FussyTree Construction

- Three phase construction
- Phase 1: Generate Frequency counts
- Phase 2: Construct “Fussy” Suffix Tree
  - Phrases and their prefixes are considered for addition only if they are ***frequent***
    - Frequency check optimization: check all substrings of candidate phrase for frequency
  - All paths in tree are thus frequent phrases
- Phase 3: Telescope paths



An Example

An Example

Doc 1	please call me asap
Doc 2	please call if you

Phase 1 : frequency tables

An Example

Doc 1	please call me asap
Doc 2	please call if you

$\longrightarrow$ (*please, call, me, asap,* -:END:-, *please, call, if, you,* -:END:-)

Phase 1 : frequency tables

An Example

Doc 1	please call me asap
Doc 2	please call if you

$\longrightarrow$ *(please, call, me, asap, -:END:-, please, call, if, you, -:END:-)* $\longrightarrow$

phrase	freq	phrase	fre
please	154	please call	14
call	46	call me	16
me	90	*me asap*	*2*
if	110	call if	6
you	184	if you	44
asap	10		

Phase 1 : frequency tables

An Example

Doc 1	please call me asap
Doc 2	please call if you

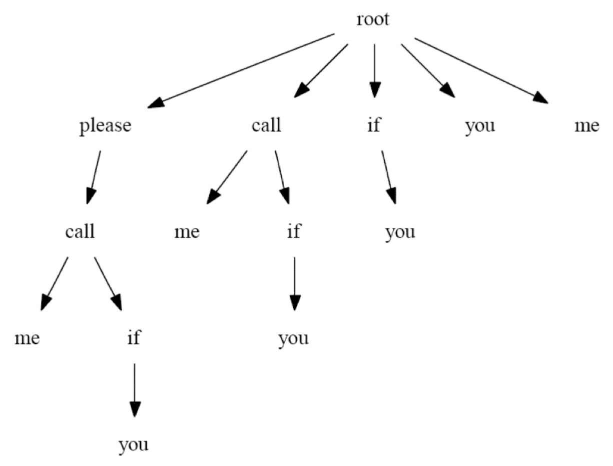
$\longrightarrow$ (*please, call, me, asap,* -:END:-, *please, call, if, you,* -:END:-) $\longrightarrow$

phrase	freq	phrase	fre
please	154	please call	14
call	46	call me	16
me	90	*me asap*	*2*
if	110	call if	6
you	184	if you	44
asap	10		

Phase 1 : frequency tables

Phase 2: Fussy Construction:

(*please, call, me, asap,* -:END:-, *please, call, if, you,* -:END:-)



isFrequent?

"please call":-

"please", "call", "please call"

An Example

Doc 1	please call me asap
Doc 2	please call if you

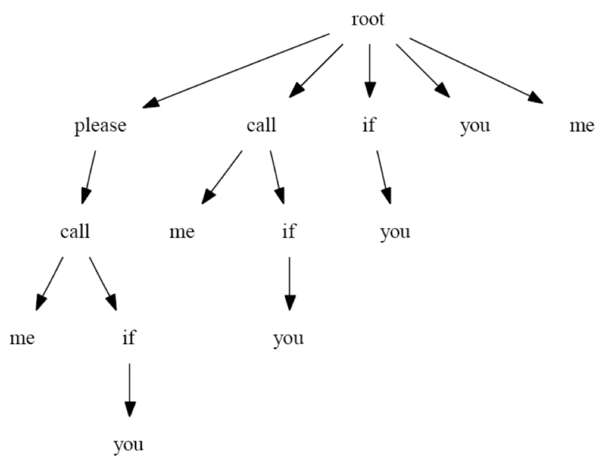
→ *(please, call, me, asap, -:END:-, please, call, if, you, -:END:-)* →

phrase	freq	phrase	fre
please	154	please call	14
call	46	call me	16
me	90	*me asap*	*2*
if	110	call if	6
you	184	if you	44
asap	10		

Phase 1 : frequency tables

Phase 2: Fussy Construction:

(please, call, me, asap, -:END:-, please, call, if, you, -:END:-)



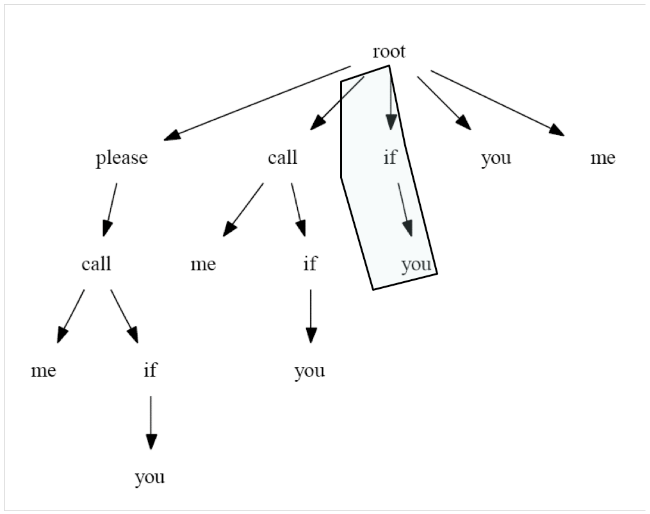
isFrequent?

"please call":-

"please", "call", "please call"

Some observations

- Problem: telescoping is not possible, variation in counts
- Often the suggested completion is too verbose / too small
- *"Please let me know if you have any problems"*
- *"Please* let me know* if you have any* problems*"*



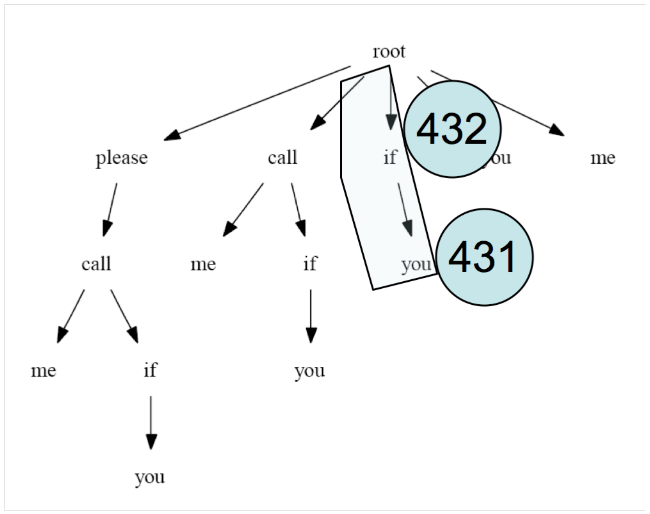
Some observations

- Problem: telescoping is not possible, variation in counts

- Often the suggested completion is too verbose / too small

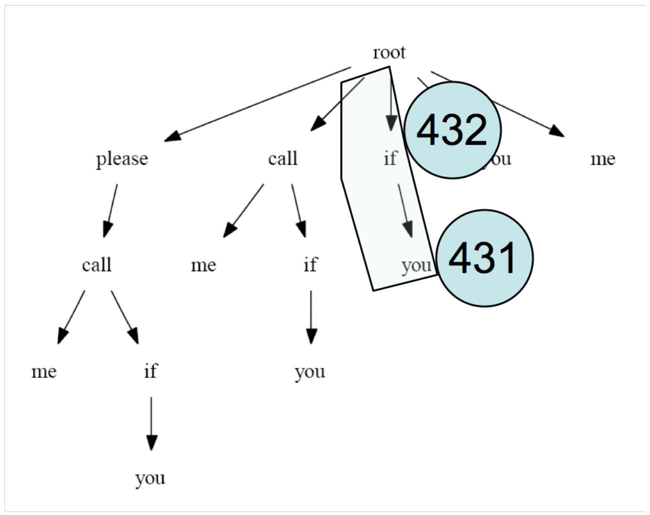
- *“Please let me know if you have any problems”*

- *“Please* let me know* if you have any* problems*”*



Some observations

- Problem: telescoping is not possible, variation in counts
- Often the suggested completion is too verbose / too small
- *"Please let me know if you have any problems"*
- *"Please* let me know* if you have any* problems*"*



Significance

- A node in the FussyTree is “significant” if it marks a phrase boundary
- Intuitively, suggestions ending on significant nodes will be better
- The FussyTree can be telescoped on significant nodes, discarding frequency counts.

Significance: definition

- phrase AB represents a **Significant node**, where:
  - *frequency*: *AB* occurs with a threshold frequency of atleast τ
  - *co-occurrence*: *AB* provides addl info over A

$$P(\text{``}AB\text{''}) > P(\text{``}A\text{''}) \cdot P(\text{``}B\text{''})$$

"let me"
"let", "me"

Significance: definition

- *comparability: AB = a factor likely as likely as A*

$$P(\text{“}AB\text{”}) \geq \frac{1}{z} P(\text{“}A\text{”})$$

"let me know this"
"let me know on"
"let me know if"

- *uniqueness: AB is much more likely than ABC*

$$P(\text{“}AB\text{”}) \geq y P(\text{“}ABC\text{”})$$

"have any problems regards"
"have any problems yours"
"have any problems john"

Significance: benefits

- No need to store counts
- Better quality results
- Telescoping possible, smaller tree size
- Process:
  - construct tree with counts
  - Scan tree for significant nodes
  - Telescope, discard counts

Online Significance

- But Significance requires an additional pass
- Why not incorporate this into the tree construction?

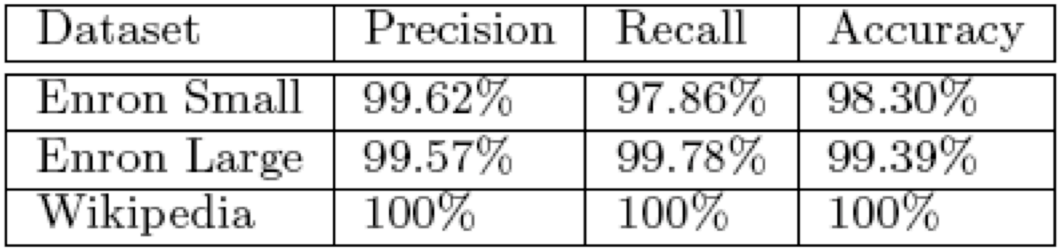
```
APPEND-TO-TREE($P$)
//align phrase to relevant node in existing suffix tree
   using cursor c, which will be last point of alignment
CONSIDER-PROMOTION($c$)
//add all the remaining words as a new path
   the cursor c is again the last point of alignment
CONSIDER-PROMOTION($c$)
CONSIDER-DEMOTION($c \rightarrow parent$)
for each child in $c \rightarrow children$
   do
      CONSIDER-DEMOTION($child$)
```

Outline

- Motivation
- Data Model
- ***Evaluation***
- Experiments
- Extensions

Online Significance

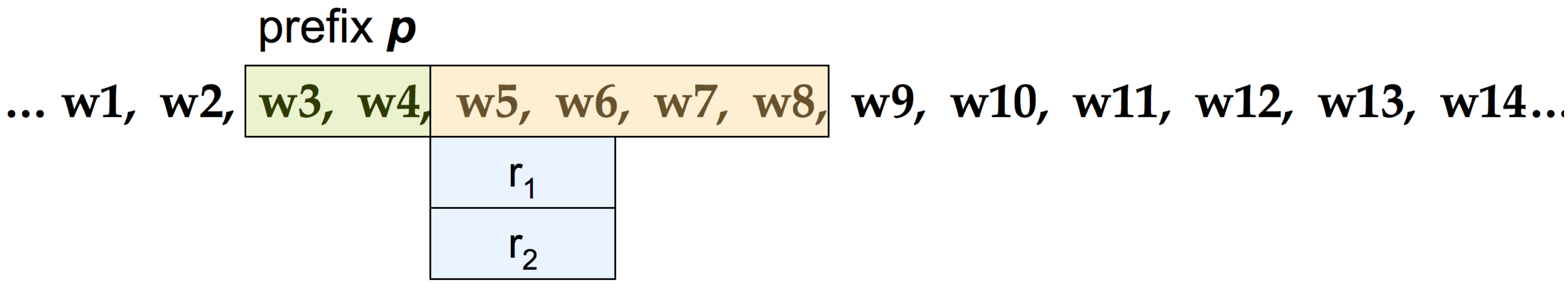
- Compare against Tree generated by *FussyTree with Offline Significance*



Dataset	Precision	Recall	Accuracy
Enron Small	99.62%	97.86%	98.30%
Enron Large	99.57%	99.78%	99.39%
Wikipedia	100%	100%	100%

Experiment Setting

- Sliding window over text, use previous text as query, future text as correct results. Do this for whole document.
- n(queries) : n(document)


prefix ***p***
… w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14…
r_1
r_2

Evaluation Metrics

- Current metrics do not support ranks:

$$Precision = \frac{\text{n(accepted completions)}}{\text{n(predicted completions)}}$$

$$Recall = \frac{\text{n(accepted completions)}}{\text{n(queries, i.e. initial word sequences)}}$$

- For ranked results:

$$Precision = \frac{\sum \text{(1/rank of accepted completion)}}{\text{n(predicted completions)}}$$

$$Recall = \frac{\sum \text{(1/rank of accepted completion)}}{\text{n(queries, i.e. initial word sequences)}}$$

Total Profit Metric

- Recall and precision do no consider length of suggestions
- Consider a “loss / profit” model, counting number of keystrokes saved

$$TPM(d) = \frac{\sum(\text{sug. length} \times \text{isCorrect}) - (\text{d} + \text{rank})}{\text{length of document}}$$

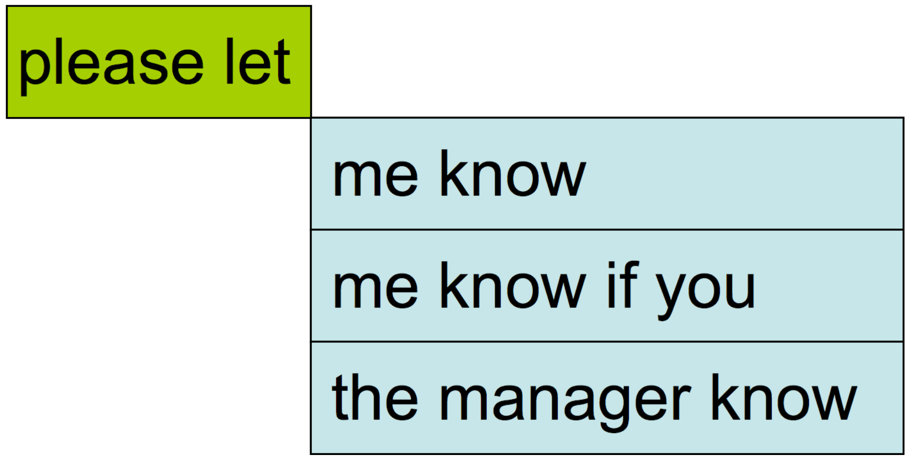
where *d* is the distraction parameter

Total Profit Metric

please let

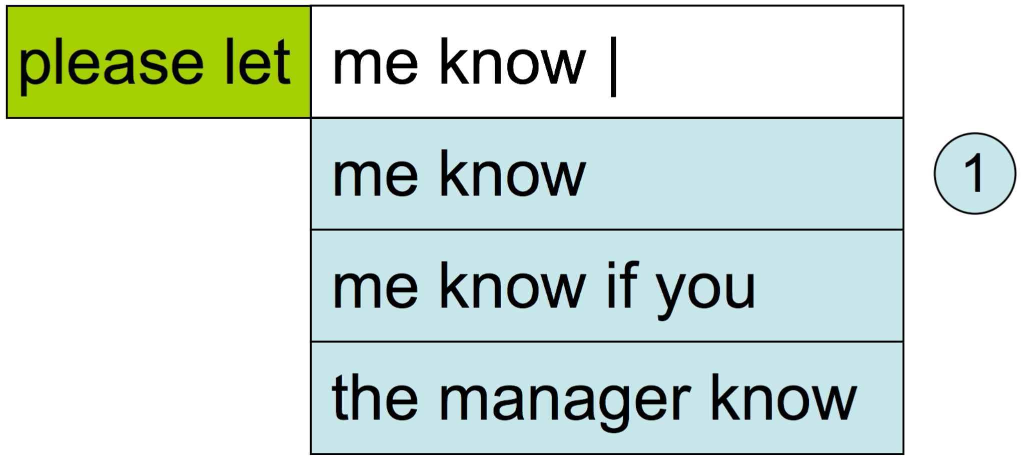
(d = 0)

Total Profit Metric



(d = 0)

Total Profit Metric



please let | me know |

me know 1

me know if you

the manager know

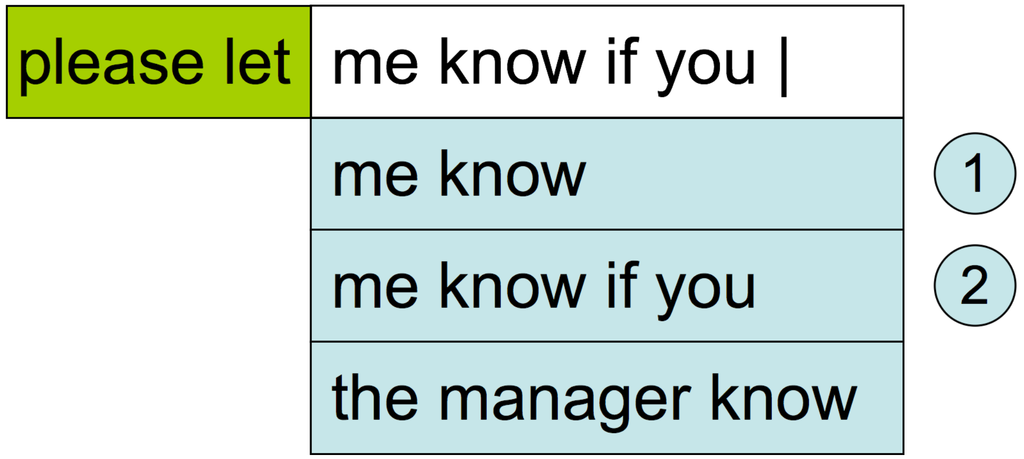
(d = 0)

Total Profit Metric

please let	me know if you |	
	me know	1
	me know if you	2
	the manager know	

(d = 0)

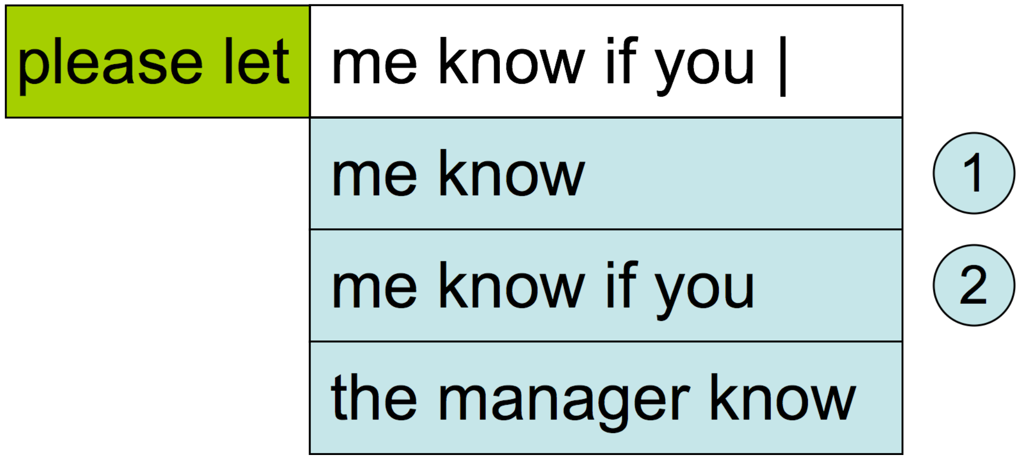
Total Profit Metric



Keys used: 2

(d = 0)

Total Profit Metric

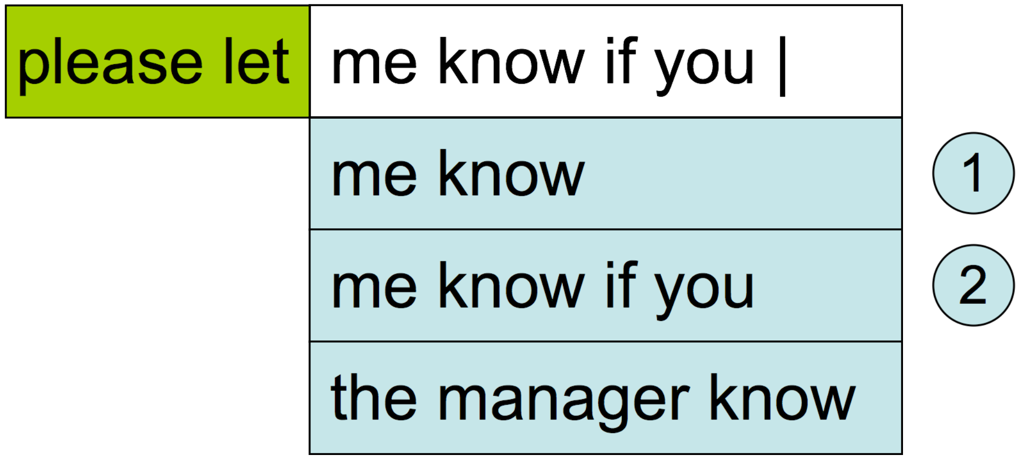


Keys used: 2

Keys saved: 14

(d = 0)

Total Profit Metric



Keys used: 2

Keys saved: 14

= 12

(d = 0)

Outline

- Motivation
- Data Model
- Evaluation
- ***Experiments***
- Extensions

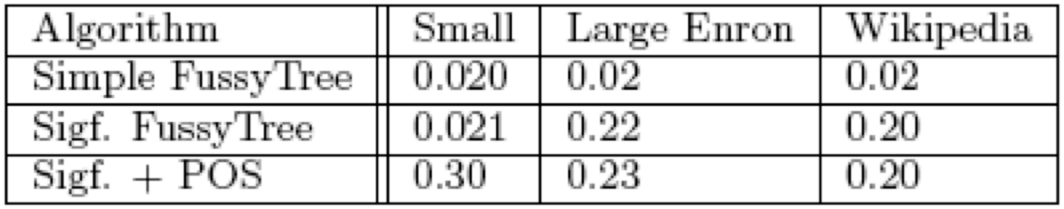
Experiments

- Multiple Corpora
  - Wikipedia Sample (40,000 documents, 53MB, large variance)
  - Enron : 1 user's "Sent" (20,842 emails / 16MB, medium variance)
  - Enron : single folder (366 emails / 250KB, less variance)

- Avg Query performance (time)
- Recall, Precision, TPM(0), TPM(1)

Prediction performance

- Autocompletion backend must be “instant”
- UI “Instantaneous” time bound = 100ms
- We are well within limit



Algorithm	Small	Large Enron	Wikipedia
Simple FussyTree	0.020	0.02	0.02
Sigf. FussyTree	0.021	0.22	0.20
Sigf. + POS	0.30	0.23	0.20

Prediction Quality

- Perspective: Dictionary-based suggestions for "Lords of The Rings" Wikipedia page
- Assume ALL named entities will be in our dictionary (anything with a wikipedia page) (226 entities)
- Using Sliding window technique, profit for **perfect** prediction is 2631 characters, or TPM(0) score is **5.99%**
  TPM(0) = no distraction penalty

Prediction Quality

Corpus: Enron Small

Dataset / Algo	Recall	Precision	TPM(0)	TPM(1)
Simple FussyTree	26.67%	80.17%	22.37%	18.09%
Sigf. FussyTree	43.24%	86.74%	21.66%	16.64%

Corpus: Enron Large

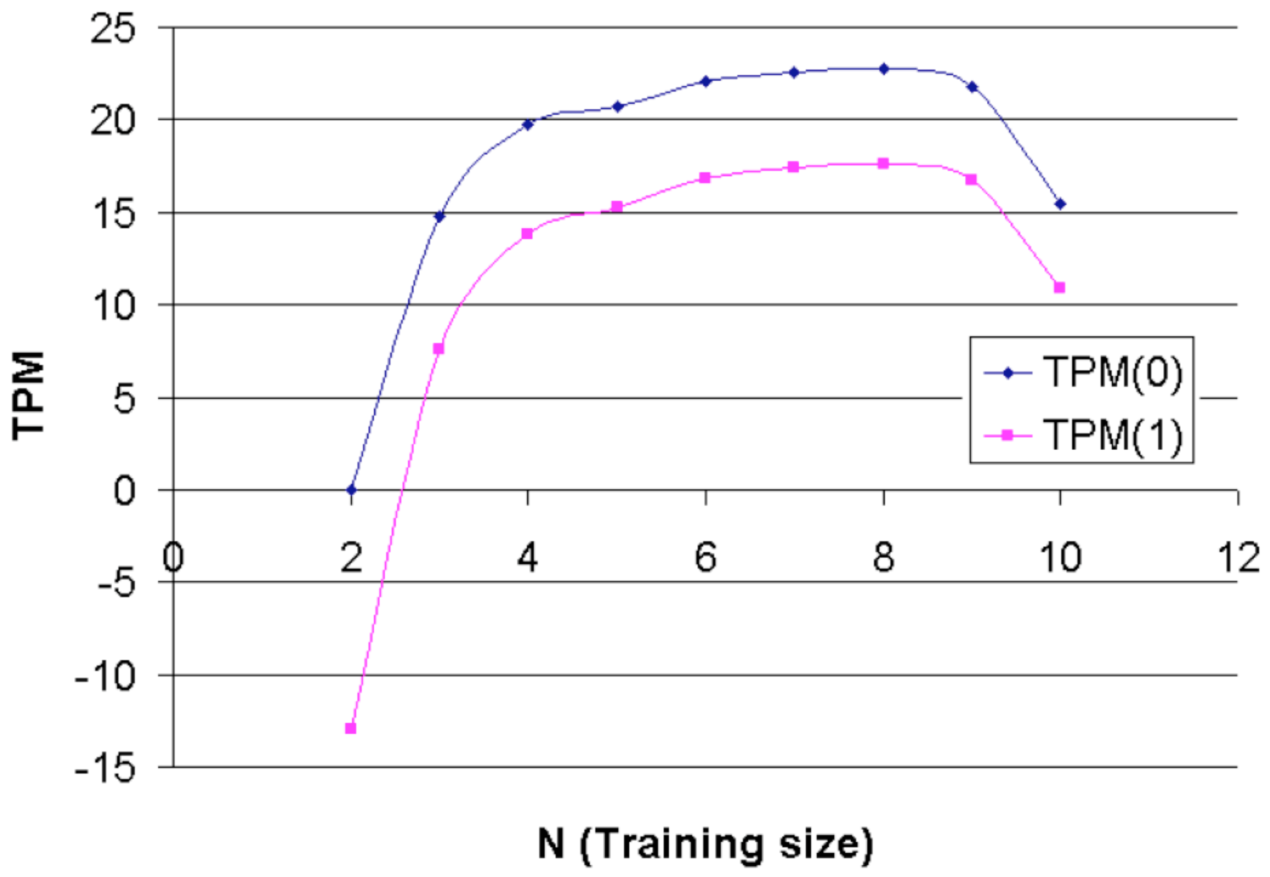
Dataset / Algo	Recall	Precision	TPM(0)	TPM(1)
Simple FussyTree	16.59%	83.10%	13.77%	8.03%
Sigf. FussyTree	26.58%	86.86%	11.75%	5.98%

Corpus: Wikipedia

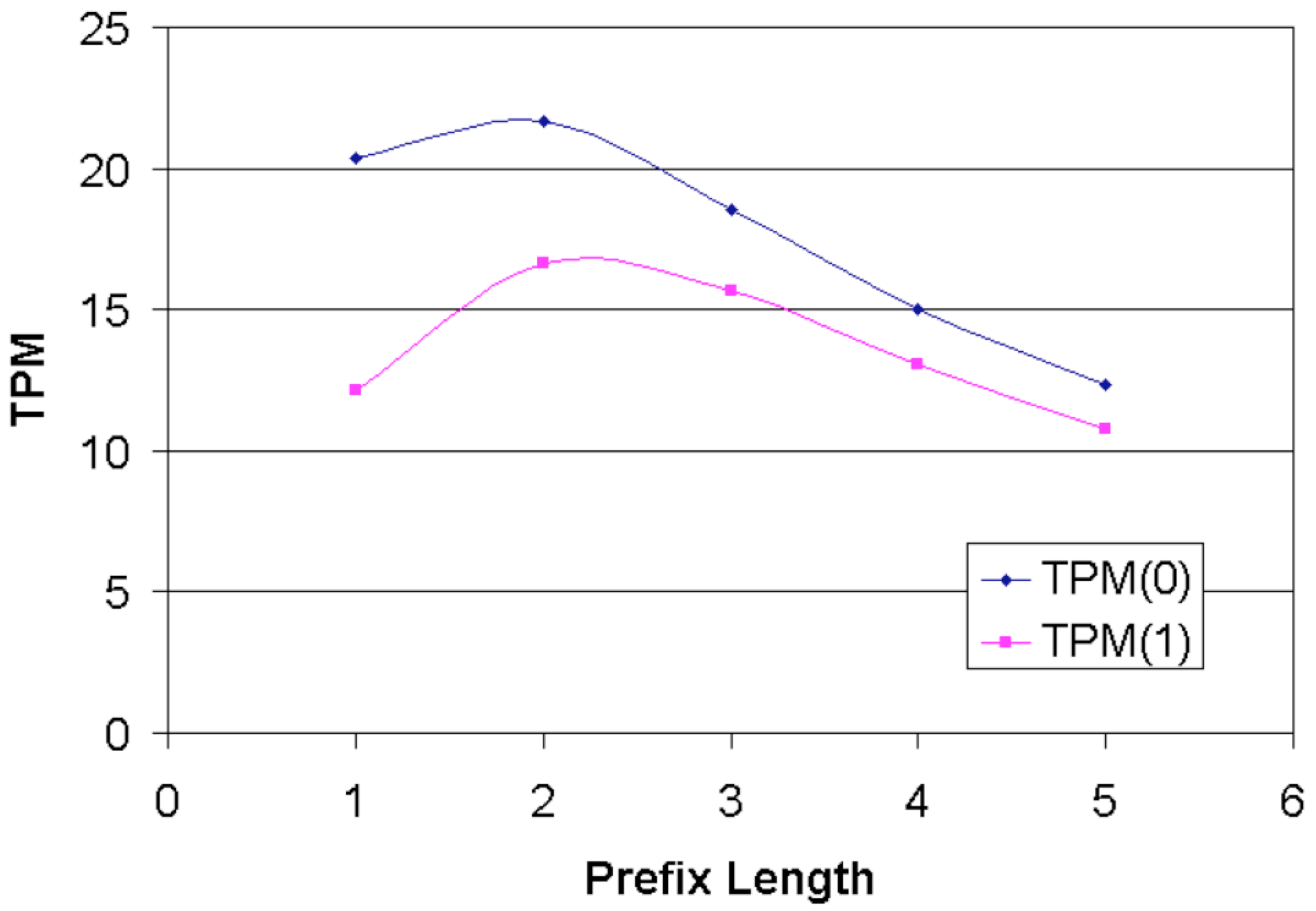
Dataset / Algo	Recall	Precision	TPM(0)	TPM(1)
Simple FussyTree	28.71%	91.08%	17.26%	14.78%
Sigf. FussyTree	41.16%	93.19%	8.90%	4.6%

- Less variance = better scores
- Significance = less TPM, better quality

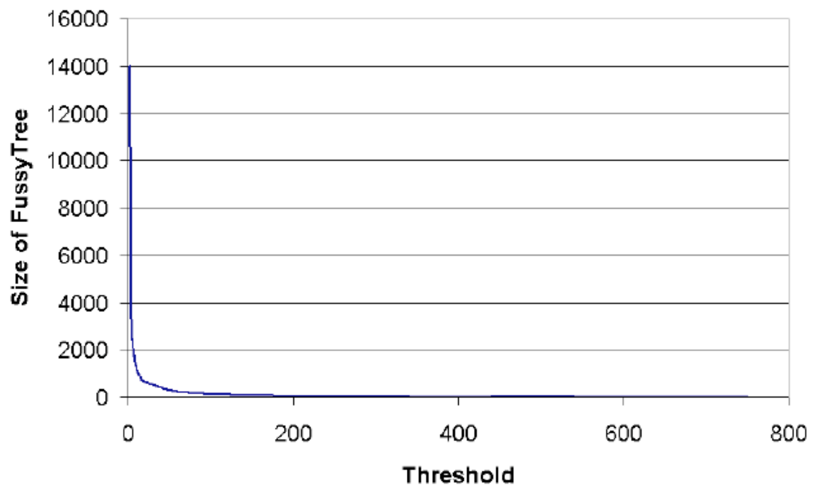
Varying training size



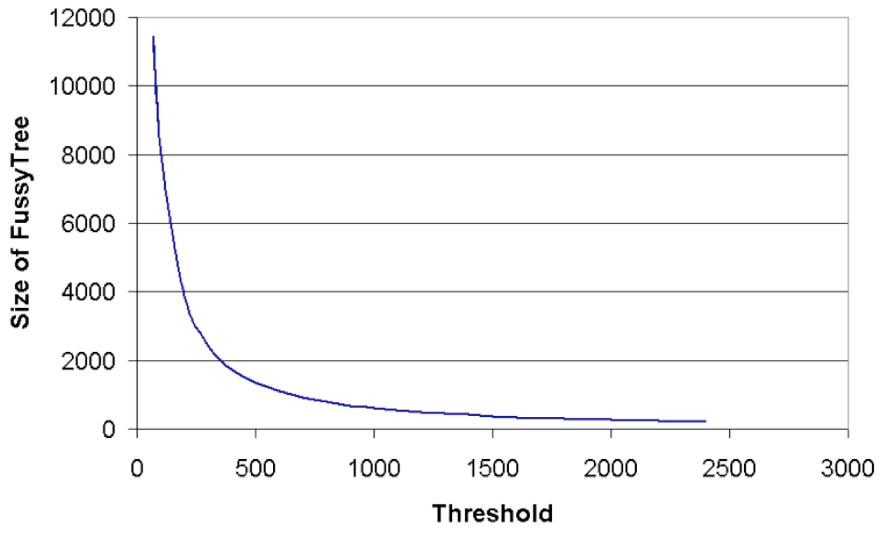
Varying prefix length



Varying Threshold



Small Enron



Large Enron

Outline

- Motivation
- Data Model
- Evaluation
- Experiments
- ***Extensions***

Possible Extensions

- Part of Speech Reranking
  - Use a POS tagger to tag phrases
  - Learn additional probability of POS-phrases
    - E.g. prob(verb-det-adj-noun) > prob(verb-det-det)
- Semantic Reranking
  - Map words to semantic meanings (Wordnet)
  - Construct a bipartite predictive model from context to suggestions
- Query Completion for structured data [Nandi, SIGMOD '07]

Related Work

- User Input Prediction
  - Motoda, *Artificial Intelligence*, 1998
  - Nevill-Manning, *Digital Libraries*, 1997
  - Bickel, *ECML*, 2005
- Suffix Tree Construction
  - Farach, *Foundations of Computer Science*, 1997
  - Tata, *VLDB*, 2004
- Frequency Estimation in Text
  - Krishnan, *SIGMOD*, 1996
  - Jagadish, *SIGMOD*, 1999

Conclusions

- Phrase level autocompletion is challenging, but can provide much greater savings beyond word-level autocompletion
- A technique to accomplish this based on “significance”
- New evaluation metrics for ranked autocompletion

Thank you!

- **http://www.eecs.umich.edu/db/usable**
- arnab@umich.edu