CSCE604135 | Perolehan Informasi (Information Retrieval)

Spelling Correction

Radityo Eko Prasojo, PhD

Content

- Wildcard queries
- Approaches for spelling correction

Wildcard Queries

- Sometimes we do not know the exact keyword that we want
  - *"Who was that US president? I know his name starts with bar..."*
- Can also be useful if the documents contain some unfixed typos, e.g. if it contains "Ba**rr**ack" instead of "Barack"

Wildcard queries: idea

- bar* → finds all docs containing words that begin with “bar”
- This is where binary trees (like B-Tree) are useful: we can retrieve all indexes in the range of **bar ≤ w < bas**
  - Remember: the indexes are sorted alphabetically!
- *bar → finding all indexes ending with “bar” is **harder**
  - Because the alphabetical order goes from left to right!
  - **Solution**: maintain another B-tree index backwards, i.e. from right to left!
- How do we process an in-between wildcard query, like “per*ent”?

Wildcard queries: performance

- Using the usual indexing mechanism, we can only enumerate all terms in the index that match the wildcard query and the corresponding postings
- E.g. "per*cent", we can lookup **per*** AND ***cent** in the B-tree, get all the union of postings for each, and intersect them
- This is a CNF → expensive!
- Solution: **permuterm index**

Permuterm Index

- Add `$` to the end of each term
- Add all possible **rotations** of the term into the B-tree index
- For example: for the term `hello`, we have:
  - `hello$`, `ello$h`, `llo$he`, `lo$hel`, `o$hell`, `$hello`, all added into the index
- Empirically, the index **quadruples** in size
  - Remember: we do not need to duplicate the postings
  - Average english words are 4.7 characters long

Permuterm Index: how is this helpful?

- We can now change any wildcard queries so that the ***** can be moved to the back!

Original query (w/o perm)	**Transformed query (w perm)**	**Example**
`X`	`X$`	`hello → hello$`
`X*`	`$X*`	`hel* → $hel*`
`*X`	`X$*`	`*llo → llo$*`
`*X*`	`X*`	`*ell* → ell*`
`X*Y`	`Y$X*`	`h*lo → lo$h*`

https://redd.it/b9t83d

Spelling Correction

- Error detection
- Error correction
  - Generate correction candidates
  - (autocorrect) pick the top 1
  - (user-centric) suggest the top-k to the users

Spelling error types

- Non-word errors
  - graffe → giraffe, yagn → yang
- Real-word errors
  - Typographical errors
    - there → three, telaah → telah
  - Cognitive errors (homophones)
    - piece → peace, too → two, your → you're, bang → bank
- Non-word errors are typically context-insensitive (easier)
- Real-world errors are almost always context-sensitive (harder)

Real-word error: need to understand the context

Non-word error: no need to understand the context clientsfromhell

Client: Do you do lemonade?

Me: Do we do… lemonade?

Client: Yes, I was told you do that here.

Me: I'm sorry, this is a graphics and print shop.

Client: I know that. I'm not an idiot.

Me: I'm sorry, I didn't mean to -

Client: Look If you can't lemonade these papers for me then I'll go somewhere else!

Me: Do you mean… laminate?

https://redd.it/eekacy

🍋 sometimes **real-word errors** are as sour as a lemonade 🍋

Trivia: **26%** of web queries contain errors (Wang et al. 2003) 📝

Trivia: **25-40%** of spelling errors are real words (Kukich, 1992) 📝

Detection

- **Non-word error:** any word that is not in the dictionary is an error!
- **Real-word error**: need to see the context
  - E.g. Flying *form* Heathrow to LAX → Flying from Heathrow to LAX
  - Checking can be done using **language model**

Correction

- Generate **candidates:** real words that are similar to the error: using **edit distance**
- Choose which one is the best: using edit distance and **language model**

Candidate generation

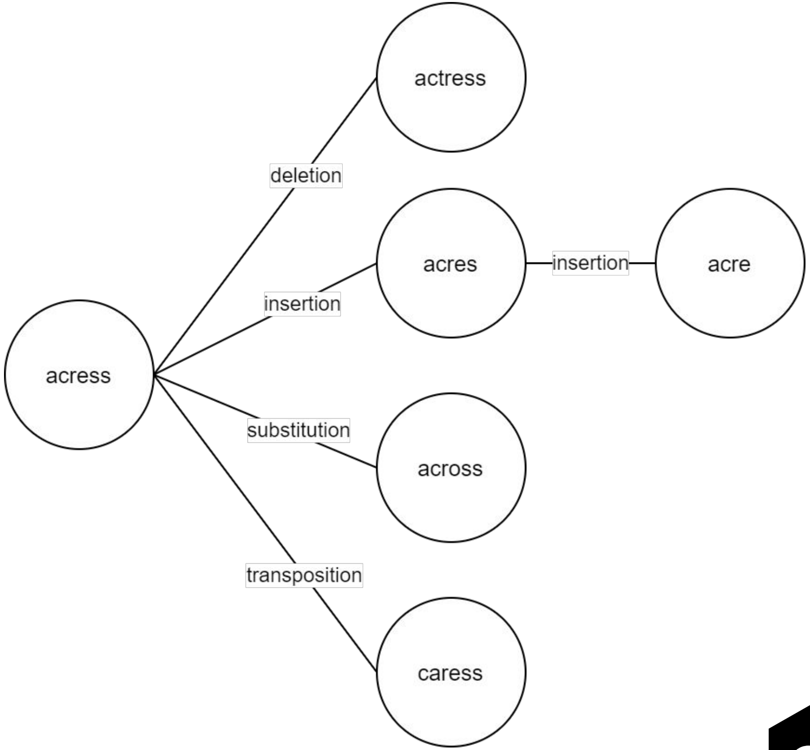
- Words with similar spellings
    - Small **edit distance** to the erroneous word
- Words with similar pronunciation
    - Small distance to the pronunciation

Levenshtein distance

- Edit distance between two strings, where edits are:
  - Insertion
  - Deletion
  - Substitution
  - Transposition between two adjacent letters
- Some people/tools count transposition and substitution as 2 edits
  - In this course we treat them all as one edit

Levenshtein distance: candidate generation

- Can be modeled as a graph
- 1 edge represents 1 edit distance
  - Acress → acre = 2 edits
- Some tools treat substitution and transposition as 2 edits
  - e.g. across and caress
- All candidates are cross-checked with a **dictionary**



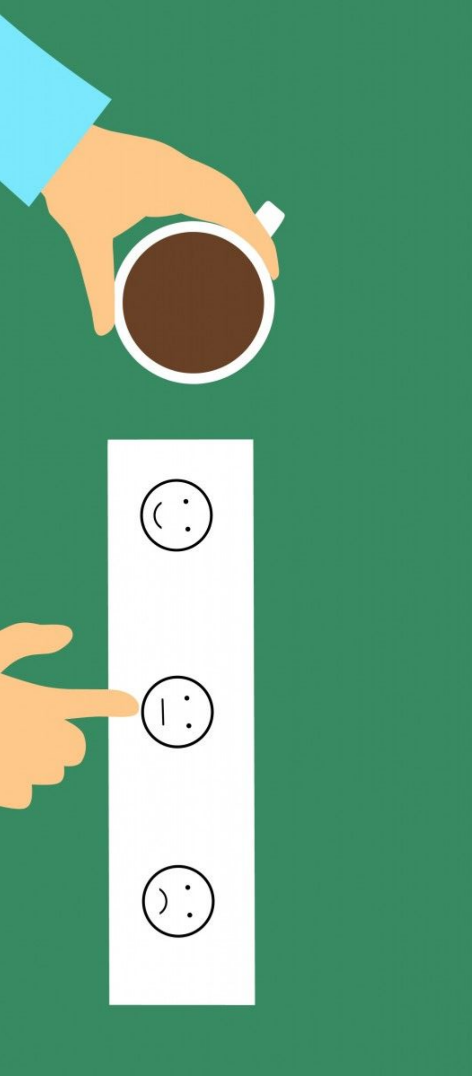
Levenshtein distance - also track what changes

Error	Candidate	Correct letter	Error letter	Type
acress	actress	t	-	deletion
acress	cress	-	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	insertion

Can also track the **char index/position** in which the change happens

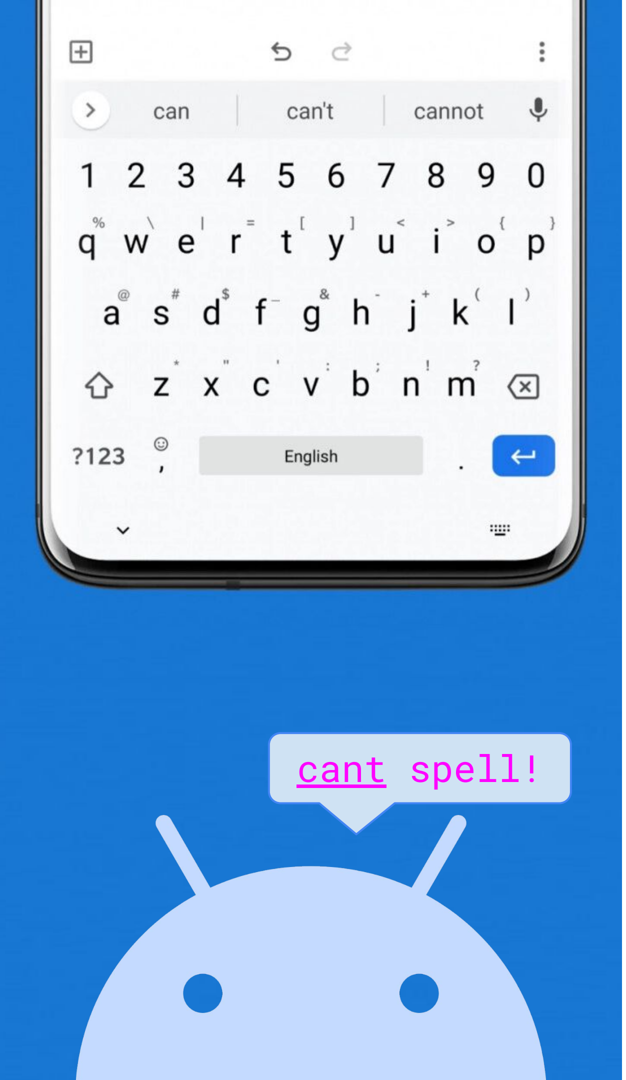
Candidate Generation

- 80% of errors are within 1 edit distance, and 99% of errors are within 2
- Also allows some non-alphanumeric character in the edits
  - E.g. spaces and - (iam → i am, data base → database, kupukupu → kupu-kupu)
  - Depending on the context or domain of the text, there can be other chars e.g. ‘@’ and ‘#’
- We can now generate all candidates within 1 edit, and then 2 edit distances → then pick the best one among them
- Note: does not guarantee the best result → a recurring theme in IR
  - We strive for good-enough results within a reasonable amount of time/space
  - e.g. in ranking best docs, implementing the best indexing mode, etc.



Picking the best candidate

- **Weighted** edit distance
- (1) from a heuristic
- (2) from a probabilistic model
- or a combination of both



Heuristic example

- Keyboard distance!
- Instead of the default 1 per edge, assign weights according to the letter distance on the keyboard
- E.g. banj → **bank** is more likely than **band**

Keyboard heuristic: weighting example

- Assuming: regular qwerty keyboard, only considering the alphabet
  - Max. horizontal distance: 9 (from Q to P)
  - Max. vertical distance: 2
  - Max. diagonal distance: sqrt(9^2+2^2) ≈ 9.75 → also maximum overall distance (z to p)
- Weight of an edit:

(max overall distance - distance)/max overall distance

- E.g. from banj to bank, the weight is (9.75 - 1)/9.75 ≈ 0.89
- From banj to band → (9.75 - 4)/9.75 ≈ 0.59
- **(self-study)** How do we incorporate insertion, deletion, and transposition?

Probabilistic Model

- Given **x** a misspelled word, and **C** a set of candidate words generated by the edit distance, find the correct word **ŵ∈C**
- **ŵ** is modeled by the following formula:

$$\hat{w} = \underset{w \in C}{\operatorname{argmax}}\, P(w|x)$$

Probabilistic Model: Bayes Rule

$$\hat{w} = \operatorname*{argmax}_{w \in C} P(w|x)$$

$$= \operatorname*{argmax}_{w \in C} \frac{P(x|w)P(w)}{\boxed{P(x)}}$$

$$= \operatorname*{argmax}_{w \in C} P(x|w)P(w)$$

x is given and is not affected by **w**, therefore *P*(**x**) is constant

Because the denominator is constant, maximizing the formula entails maximizing the **numerator**

P(**x**|**w**) = probability of getting a typo **x** given a real word **w**

P(**w**) = probability of seeing a real word **w**

Getting *P*(**w**): unigram language model

- Given a collection of documents / a corpus, tokenize all the words, let the total number of words be **T**, and let *C*(**w**)=the number of occurrences of the word **w** in the corpus

$$P(w) = \frac{C(w)}{T}$$

- “Unigram” because it involves only one word/token as parameter

Unigram language model: example

- Counts from 404,253,213 words in Corpus of Contemporary English (COCA)

Word (**w**)	Frequency of word	*P*(**w**)
actress	9,321	.0000230573
cress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
across	120,844	.0002989314
acres	12,874	.0000318463

Getting $P(x|w)$: create confusion "matrix"

- From a large corpus, detect all the typos
- Take a sample from the detected typos, then **manually** fix them
- Count the number of fixes being made
  - `del[x,y]: count(xy typed as x)`
  - `ins[x,y]: count(x typed as xy)`
  - `sub[x,y]: count(y typed as x)`
  - `trans[x,y]: count(xy typed as yx)`
- Make a confusion "matrix"

Confusion matrix example

[Kernighan, Church, & Gale, 1990]

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

$$P(x|w) = \begin{cases} \dfrac{\mathrm{del}[w_{i-1}, w_i]}{\mathrm{count}[w_{i-1} w_i]}, & \text{if deletion} \\[2ex] \dfrac{\mathrm{ins}[w_{i-1}, x_i]}{\mathrm{count}[w_{i-1}]}, & \text{if insertion} \\[2ex] \dfrac{\mathrm{sub}[x_i, w_i]}{\mathrm{count}[w_i]}, & \text{if substitution} \\[2ex] \dfrac{\mathrm{trans}[w_i, w_{i+1}]}{\mathrm{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Smoothing $P(\mathbf{x}|\mathbf{w})$

- Because we rely on sampling, some cases are bound to be left out
  - E.g. there might be typos of ‘a’ written as ‘q’, but our sample gets 0 of them
- For these cases, the normal $P(\mathbf{x}|\mathbf{w})$ formula returns 0, which seems unrealistic
- A simple solution is to add 1 to all counts and then if there is a $|A|$ character alphabet, to normalize appropriately.
  - E.g. for substitution:

$$P(x \mid w) = \frac{\mathrm{sub}[x,w]+1}{\mathrm{count}[w]+A}$$

Example: acress

Candidate	Correct letter	Error letter	Type	*P*(x|w)	*P*(w)	*P*(x|w)**P*(w)*10^9
actress	t	-	del	.000117	.0000230573	2.7
cress	-	a	ins	.00000144	.0000005442	.00078
caress	ca	ac	trans	.00000164	.0000016969	0.0028
access	c	r	subs	.000000209	.0000916207	0.19
across	o	e	subs	.0000093	.0002989314	2.8
acres	-	s	ins	.0000321	.0000318463	1.0

Evaluation - test sets

- Wikipedia's list of common English misspelling
- Aspell filtered version of that list
- Birkbeck spelling error corpus
- Peter Norvig's list of errors (includes Wikipedia and Birkbeck, for training or testing)

Real-word spelling errors

- `…leaving in about fifteen` **`minuets`** `to go to her house.`
- `The design` **`an`** `construction of the system…`
- `Can they` **`lave`** `him my messages?`
- `The study was conducted mainly` **`be`** `John Black`

- Detecting and correcting these words require understanding the context → **contextual language model**

Contextual Language Model

- As opposed to the **unigram model**, contextual model involves multiple words
- Example: n-gram language model, with n>1

n-gram language model

- Given a collection of documents / a corpus, tokenize all the words
- For any n given words, we define a probability

$$P(\mathbf{w_n} | \mathbf{w_{n-1}}, \mathbf{w_{n-2}}, \ldots, \mathbf{w_1})$$

  by counting the word occurrences after tokenization
- E.g. if we set n=3, given words “saya makan” (e.g. occur 100 times), we can count how many times “nasi” appears after that (e.g. 30 times)
  - P(**nasi**|**makan**,**saya**) = 0.3
- *P*(**nasi**|**makan**,**saya**) should be higher than *P*(**air**|**makan**,**saya**)

n-gram language model

- For any sequence of words of length q > n, we can compute the probability of such a sequence occurs as:

$$P(\mathbf{w}_1 \ldots \mathbf{w}_q) = P(\mathbf{w}_1) P(\mathbf{w}_n | \mathbf{w}_{n-1}, \mathbf{w}_{n-2}, \ldots, \mathbf{w}_1) P(\mathbf{w}_{n+1} | \mathbf{w}_n, \mathbf{w}_{n-1}, \ldots, \mathbf{w}_2) \ldots P(\mathbf{w}_q | \mathbf{w}_{q-1}, \mathbf{w}_{q-2}, \ldots, \mathbf{w}_{q-n+1})$$

- For example, if n = 2, then

$$P(\mathbf{w}_1 \ldots \mathbf{w}_q) = P(\mathbf{w}_1) P(\mathbf{w}_2 | \mathbf{w}_1) P(\mathbf{w}_3 | \mathbf{w}_2) \ldots P(\mathbf{w}_q | \mathbf{w}_{q-1})$$

- And for “saya makan nasi goreng”:

$$P(\textbf{saya makan nasi goreng}) = P(\textbf{saya}) P(\textbf{makan} | \textbf{saya}) P(\textbf{nasi} | \textbf{makan}) P(\textbf{goreng} | \textbf{makan})$$

n-gram language model: smoothing

- Some word sequence might never appear → the probability is then 0
  - E.g. "makan bata" in bigram language model
  - But this might just be because our corpus is not extensive enough
- 1-smoothing: similar to the bayes rule before:
  - from **original** P(bata|makan) = count(makan bata)/count(makan)
  - to **1-smoothed** P(bata|makan) = (count(makan bata)+1)/(count(makan)+|Words|)
- Or, using **unigram interpolation**
  - $P(bata|makan) = \lambda P_{unigram}(bata) + (1-\lambda)P_{original}(bata|makan)$
  - claimed to perform better than the 1-smoothing

Logarithm to handle very small value

- These probabilities can be very small → underflow risk
- Use log to work with bigger number
- For example, if n = 2, then

$$P(\mathbf{W}_1 \ldots \mathbf{W}_q) = P(\mathbf{W}_1)\,P(\mathbf{W}_2 | \mathbf{W}_1)\,P(\mathbf{W}_3 | \mathbf{W}_2) \ldots P(\mathbf{W}_q | \mathbf{W}_{q-1})$$

$$\log(P(\mathbf{W}_1 \ldots \mathbf{W}_q)) = \log(P(\mathbf{W}_1)) + \log(P(\mathbf{W}_2 | \mathbf{W}_1)) + \log(P(\mathbf{W}_3 | \mathbf{W}_2)) + \ldots + \log(P(\mathbf{W}_q | \mathbf{W}_{q-1}))$$

n-gram model to detect real-word errors

- Set a probability **threshold**, then run a given sentence into the n-gram model and compute the probability. If it is below the threshold, then we suspect it has some errors.

`Can they` **`lave`** `him my messages?`

- Suppose we use bigram, compute P(they|can), ..., P(messages|my), if P(lave|they) and P(him|lave) are below threshold, **lave** is an error
- A good threshold can be obtained via training (machine learning)

Using contextual model to pick the best fix

- Similar to before: generate candidates using edit distance
- Next, instead of using a unigram language model, we look at the context by using a contextual language model, for example **bigram**

"a stellar and versatile **acress** whose combination of sass and glamour..."

- From the Coca corpus with 1-smoothing
    - P(actress|versatile)=.000021, P(whose|actress) = .0010
      P(across|versatile) =.000021, P(whose|across) = .000006
      P("versatile **actress** whose") = .000021*.0010 = 210 $\times 10^{-10}$
      P("versatile **across** whose") = .000021*.000006 = 1 $\times 10^{-10}$

Improvements to the discussed models

- **Bidirectional model**: though most of the times reading is left-to-right, some contextual information can be obtained from right to left
    - E.g.: “Before **he** became president, **Obama** …”
- **Pronunciations**: e.g. by transforming texts into **phonemes**
    - Phoneme of “live” is /lɪv/, whereas “leave” is /liv/, cutting the edit distance from 2 to 1
- **Incorporate other heuristics** like the keyboard into the probabilistic model

Supported by the Kampus Merdeka grant of
Ministry of Education, Culture, Research, and Technology
of Republic of Indonesia