SE 481 Introduction to Information Retrieval

Module #4 — Spell Collections

Passakorn Phannachitta, D.Eng.
passakorn.p@cmu.ac.th
College of Arts, Media and Technology
Chiang Mai University, Chiangmai, Thailand

Agenda

- Spelling correction

Spelling correction

- A.K.A. Auto collection

Ref: https://techalook.com/how-to/turn-on-off-auto-correct-samsung/

Rate of spelling errors

- Of course, it depends on environments and the application
- In general, error rate is around 1% - 20%
- Higher in some environments — phone-sized keyboard
- Higher in some applications — web queries
- Higher in some language — vowels and tone marks exceptionally increase error rate for Thai language

Tasks

- Spelling error detection
- Spelling error correction
  - Auto correct
    (replace an unknown word with the known one — without notice)
  - Suggest a correction
    (suggest a known word when an unknown word is detected)
  - Suggest list of possible corrected words
    (show a list of known words with the k-highest probability)

Type and difficulties

- Non-word errors
  - sofware -> software

- Real word errors
  - Typographical errors

    three -> there
  - Cognitive errors (i.e., homophones)

    too -> two

- The main difference is the context sensitivities

Non-word spelling errors

- **Principle** — words **not** in a predefined **dictionary** is an error
- The larger the dictionary the better (assume we have an unlimited computing resources)
- Noisy sources are not considered as a good dictionary

- **Approach** — Generate the list of word **candidates** and **choose** the most comprehensible one
  - Candidate generation
  - Scoring and ranking

Real word & non-word spelling errors

- For each word *w*, generate candidate set:
  - Find candidate words with similar **pronunciations** and/or
  - Find candidate words with similar **spellings** and/or
  - Also Include *w* in candidate set

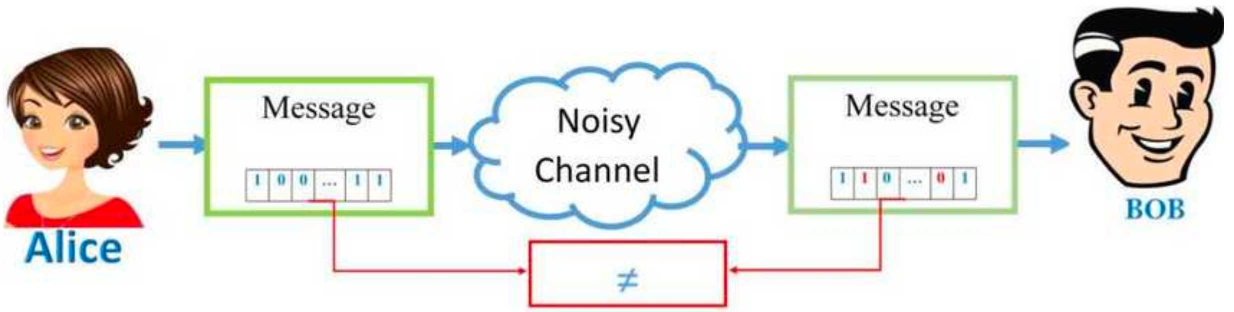
- Choose best candidate
  - Noisy Channel view of spell errors
  - Consider whether the surrounding words **makes any sense**

  - Flying **form** CNX to BKK -> Flying **from** CNX to BKK

Noisy channel model of spelling

- Assume terms are independent

Ref: https://slideplayer.com/slide/4848544/



Ref: https://www.researchgate.net/figure/Figure-1-Communication-over-Noisy-Channel-Error-correcting-codes-harnesses-the-coding_fig1_318468338

Unit

- Character bigrams (commonly referred to as k-grams)
  - e.g., good morning — $g go oo od d$ $m mo or rn ni in ng g$
  - $ is a word break notation.

- Word bigrams (commonly referred to as n-grams)
  - e.g., MongoDB is a non-relational database —
    MongoDB is | is a | a non-relational | non-relational database

Noisy channel and Bayes' rule

- We see an observation *x* of a misspelled word
- Find the correct word w'

$$
\begin{aligned}
w' &= \underset{w \in V}{argmax}\ P(w|x) \\
&= \underset{w \in V}{argmax}\ \frac{P(x|w)P(x)}{P(x)} \\
&= \underset{w \in V}{argmax}\ P(x|w)P(w)
\end{aligned}
$$

Bayes

Noisy channel model

prior

Non-word spelling error example — e.g.,

defet

Candidate generation

- Words with similar spelling
  - Small **edit distance** to error

- Words with similar pronunciation
  - Small distance of pronunciation to error

Candidate testing

- Damerau-Levenshtein edit distance — Minimal edit distance between two string, where edits are:
  - Inserting
  - Deletion
  - Substitution
  - Transposition of two adjacent letters

- https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

Word within distance = 1 of defet

Error	Candidate correction	Correct letter	Error letter	Type
defet	defeat	a	-	deletion
defet	decet	c	f	substitution
defet	defect	c	-	deletion
defet	deft	-	e	Insertion
defet	defer	r	t	substitution
defet	Deeft	ef	fe	transposition

Candidate generation

- Observation
  - 80% of errors are within edit distance = 1
  - ~100% of errors are within edit distance = 2
  - The first letter is rarely a typo
- Insertion also includes space or hyphen
  - thismethod -> this method
- Deletion also includes space
  - data base -> database

Candidate generation — procedure

1. Run through the dictionary and calculate the edit distance between the query and the dictionary word
2. List all those within the edit distance ≤ 2

How to rank this resultant list —> use IR

Candidate generation — procedure

$$
\begin{aligned}
w' &= \underset{w \in V}{argmax}\ P(w|x) \\
&= \underset{w \in V}{argmax}\ \frac{P(x|w)P(x)}{P(x)} \\
&= \underset{w \in V}{argmax}\ P(x|w)P(w)
\end{aligned}
$$

We need to know P(w)

Language model

- We can estimate *P(w)* if we have access to a very large supply of words, i.e., corpus, that can represent the **context**

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

  - *C(w)* is the number of occurrence of *w in the* corpus*, and T* is the total number of terms in the corpora

- Counting !

Example

- Let's try the Corpus of Contemporary English (COCA — https://www.english-corpora.org/coca/) as the corpus
- At of the end of 2019, T ~ 1,000,000,000

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

Word	Frequency	*P(w)*	Rank
defeat	21,947	0.000021947	1
decet	6	0.000000006	5
defect	3,973	0.000003976	2
deft	1,240	0.000001240	4
defer	2,239	0.000002239	3
Deeft	0	0	6

```
COCA = pd.DataFrame([['defeat',21947],['decet',6],['defect',3973],['deft',1240],['defer',22
['Deeft',0]],columns=['word','frequency'])
COCA_pop = 1e9
COCA['P(w)'] = COCA['frequency']/COCA_pop
COCA['rank'] = COCA['frequency'].rank(ascending=False).astype(int)
```

Example

- Let's change the corpus to Wikipedia (https://www.english-corpora.org/wiki/)
- T ~ 1,900,000,000

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

Word	Frequency	*P(w)*	Rank
defeat	121,408	0.00006389894737	1
decet	81	0.00000004263157	5
defect	7,793	0.00000410157894	2
deft	814	0.00000042842105	4
defer	1,416	0.00000074526315	3
Deeft	0	0	6

Same

```
WIKI = pd.DataFrame([['defeat',121408],['decet',81],['defect',7793],['deft',814],['defer',1416],
['Deeft',0]],columns=['word','frequency'])
WIKI_pop = 1.9e9
WIKI['P(w)'] = WIKI['frequency']/WIKI_pop
WIKI['rank'] = WIKI['frequency'].rank(ascending=False).astype(int)
```

Example

- If we change the corpus to IULA Spanish-English Technical Corpus (https://repositori.upf.edu/handle/10230/20052)
- T ~ 2,100,000

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

Word	Frequency	*P(w)*	Rank
defeat	11	0.0000052	2
decet	0	0	4
defect	180	0.0000857	1
deft	0	0	4
defer	11	0.0000052	2
Deeft	0	0	4

Corpus does matter

```
IULA = pd.DataFrame([['defeat',11],['decet',0],['defect',180],['deft',0],['defer',11],
['Deeft',0]],columns=['word','frequency'])
IULA_pop = 2.1e6
IULA['P(w)'] = IULA['frequency']/IULA_pop
IULA['rank'] = IULA['frequency'].rank(ascending=False).astype(int)
```

Channel model probability

- *P(x|w)* = probability of the edit
  - deletion | insertion | substitution | transposition

- Misspelled word $x = x_1, x_2, x_3, ..., x_m$
- Correct word $w = w_1, w_2, w_3, ..., w_m$

Channel model probability

- *Kernighan, Church, Gale 1990*

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

Channel model probability

- Consult the collected list of errors, e.g., Peter Norvig’s collections http://norvig.com/ngrams/
- Note — we cannot model unseen errors
- count_1edit.txt

```
norvig = pd.read_csv('http://norvig.com/ngrams/count_1edit.txt',sep='\t',encoding = "ISO-8859-1",header=None)
norvig.columns = ['term', 'edit']
norvig = norvig.set_index('term')
print(norvig.head())
```

```
      edit
term
e|i    917
a|e    856
i|e    771
e|a    749
a|i    559
```

Channel model probability

- Then the correction notes

- count_big.txt

```
norvig_orig = pd.read_csv('http://norvig.com/ngrams/count_big.txt',sep='\t',encoding = "ISO-8859-1",header=None)
norvig_orig = norvig_orig.dropna()
norvig_orig.columns=['term','freq']
print(norvig_orig.head())
```

	term	freq
0	a	21160
1	aah	1
2	aaron	5
3	ab	2
4	aback	3

P(x|w)

Candidate correction	Correct letter	Error letter	x|w	P(x|w)	
defeat	a	-	e | ea	354 / 27,583	0.012833
decet	c	f	f | c	4 / 144,964	0.000028
defect	c	-	e | ec	47 / 14,686	0.003167
deft	-	e	e | _	2 / 116,374	0.000003
defer	r	t	t | r	11 / 444,459	0.000036
Deeft	ef	fe	fe | ef	21 / 9,689	0.003311

P(x|w)

```python
with mp.Pool(processes=8) as pool:
    freq_list = pool.map(functools.partial(get_count, norvig_orig=norvig_orig), character_set)

freq_df = pd.DataFrame([character_set, freq_list], index=['char', 'freq']).T
freq_df = freq_df.set_index('char')

COCA['P(x|w)'] = [(norvig.loc['e|ea'].values / freq_df.loc['ea'].values)[0],
    (norvig.loc['f|c'].values / freq_df.loc['c'].values)[0],
    (norvig.loc['e|ec'].values / freq_df.loc['ec'].values)[0],
    (norvig.loc['e| '].values / freq_df.loc['e'].values)[0],
    (norvig.loc['t|r'].values / freq_df.loc['r'].values)[0],
    (norvig.loc['fe|ef'].values / freq_df.loc['ef'].values)[0]]
```

$P(x|w)P(w)$ — Using COCA

Candidate correction	$P(w)$ COCA	P(x|w)	10^9 P(x|w)P(w)
defeat	0.000021947	0.012833	281.6676
decet	0.000000006	0.000028	0.0001
defect	0.000003976	0.003167	12.5821
deft	0.000001240	0.000003	0.0039
defer	0.000002239	0.000036	0.0795
Deeft	0	0.003311	0.0000

```
COCA['109 P(x|w)P(w)'] = 1e9 * COCA['P(w)'] * COCA['P(x|w)']
```

$P(x|w)P(w)$ — Using IULA

Candidate correction	$P(w)$ IULA	P(x|w)	10^9 P(x|w)P(w)
defeat	0.0000052	0.012833	67.2256
decet	0	0.000028	0.0000
defect	0.0000857	0.003167	271.4487
deft	0	0.000003	0.0000
defer	0.0000052	0.000036	0.1861
Deeft	0	0.003311	0.0000

```
IULA['P(x|w)'] = COCA['P(x|w)']
IULA['109 P(x|w)P(w)'] = 1e9 * IULA['P(w)'] * IULA['P(x|w)']
```

Executed in parallel

```python
COCA, WIKI, IULA = gen_table()
norvig, norvig_orig = read_norvig()
character_set = list(map(''.join, itertools.product(ascii_lowercase, repeat=1))) + list(map(''.join, itertools.product(ascii_lowercase, repeat=2)))

with mp.Pool(processes=8) as pool:
    freq_list = pool.map(functools.partial(get_count, norvig_orig=norvig_orig), character_set)

freq_df = pd.DataFrame([character_set, freq_list], index=['char', 'freq']).T
freq_df = freq_df.set_index('char')

COCA['P(x|w)'] = [(norvig.loc['e|ea'].values / freq_df.loc['ea'].values)[0],
    (norvig.loc['f|c'].values / freq_df.loc['c'].values)[0],
    (norvig.loc['e|ec'].values / freq_df.loc['ec'].values)[0],
    (norvig.loc['e| '].values / freq_df.loc['e'].values)[0],
    (norvig.loc['t|r'].values / freq_df.loc['r'].values)[0],
    (norvig.loc['fe|ef'].values / freq_df.loc['ef'].values)[0]]

COCA['109 P(x|w)P(w)'] = 1e9 * COCA['P(w)'] * COCA['P(x|w)']

IULA['P(x|w)'] = COCA['P(x|w)']
IULA['109 P(x|w)P(w)'] = 1e9 * IULA['P(w)'] * IULA['P(x|w)']
```

What we did?

- **Corpus** — tells the estimated appearance probability for each word.

- **Misspelt statistics** — tell the estimated misspelt frequency

- Multiplication of the two components tells which known words are more likely the correction of the misspelt word.

Other source for the Misspelt statistics

- http://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines
- http://aspell.net/test/
- http://www.ota.ox.ac.uk/headers/0643.xml

Context-sensitive spelling correction

- Nowadays, Event driven programming has become a **domnant** programming paradigm.
- Anything **taht** can go wrong will go wrong.
- Modern commodity computers are **equippped** with multicore CPUs.
- It is difficult to make a **defet**-free software product.

Researches say 25% - 40% of spelling errors are real words

Approach

- For each word in sentence (phrase, query …)
  - Generate the candidate set
    - The word itself
    - All known words with edit distance = 1
    - All known words that are homophones
  - Choose best candidates with the Noisy channel model

The Noisy channel model is somewhat modified.

Noisy channel for real-word spell correction

- Given a sentence $x_1, x_2, x_3, \ldots, x_n$
- Generate a set of candidates for each word x_i

$$\text{Candidate}(x_1) = \{x_1, w_1, w_1', w_1'', \ldots\}$$
$$\text{Candidate}(x_2) = \{x_2, w_2, w_2', w_2'', \ldots\}$$
$$\text{Candidate}(x_3) = \{x_3, w_3, w_3', w_3'', \ldots\}$$

- Choose the sequence W that maximize $P(W|x_1, x_2, \ldots, x_n)$

$$w' = \underset{w \in V}{argmax}\ P(x|w)P(w)$$

Incorporating context

- If we do not have access to the preferred context-specific corpus,
  - Determining whether **defeat** or **defect** is more appropriate will require looking at the context words

- There are better language models, simplest ones are such as **bigram language model —** look back just one previous word

$$P(w_1...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_{n-1})$$

  - Counting the co-occurrences divided by occurrences

Using a bigram language model

- “It is difficult to make a **defet**-free software product.”
- Let’s just use the COCA

$P(w_k \mid w_{k-1})$	$C(w_{k-1} \mid w_k) / C(w_{k-1})$	Evaluate	
P(defeat | a)	C(a defeat) / C(defeat)	607 / 21,947	0.02765
P(defect | a)	C(a defect) / C(defect)	453 / 3,976	0.11393
P(free | defeat)	C(defeat free) / C(free)	1 / 256,258	0.000004
P(free | defect)	C(defect free) / C(free)	5 / 256,258	0.000020

- *P(“a defeat free”)* = 0.02765 x 0.000004 = 0.0000001
- P(“a defect free”) = 0.11393 x 0.00002 = 0.0000022

Incorporating context

- Choice of corpora generates less affect when contexts are considered.

Improved the edit distance component

- The basic unit is extended **pronunciation** to words
- In assumed noise-free channel, simply compute a distance function that can represent sound and find the word with minimum distances

Noteworthy algorithm — Soundex

function SOUNDEX(*name*) **returns** *soundex form*

1. Keep the first letter of *name*
2. Drop all occurrences of non-initial a, e, h, i, o, u, w, y.
3. Replace the remaining letters with the following numbers:

   b, f, p, v $\rightarrow$ 1
   c, g, j, k, q, s, x, z $\rightarrow$ 2
   d, t $\rightarrow$ 3
   l $\rightarrow$ 4
   m, n $\rightarrow$ 5
   r $\rightarrow$ 6

4. Replace any sequences of identical numbers, only if they derive from two or more letters that were *adjacent* in the original name, with a single number (e.g., 666 $\rightarrow$ 6).
5. Convert to the form `Letter Digit Digit Digit` by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

Check — http://sites.rootsweb.com/~nedodge/transfer/soundexlist.htm

Noteworthy algorithm — Soundex

SOUNDEX CALCULATOR

Original work and documentation by Moishe Miller
expanded to allow a list of names by Renee Bunck

This form will convert a surname to the corresponding soundex code (4 characters), using the rules specified in the National Archive's handbook. Type the name(s) in the **ENTRY** box and click the **Calculate Soundex** button. The Soundex code(s) will be displayed in the **RESULT** box.

Enter a Surname or list of Surnames separated by commas:

Programming

Calculate Soundex

Results: P626

Note: Requires JavaScript 1.0 capable browser.

SOUNDEX CODING GUIDE

The number	Represents the letters
1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

Disregard the letters A, E, I, O, U, W, Y, and H.

Check — http://sites.rootsweb.com/~nedodge/transfer/soundexlist.htm

Applied to noisy channel

- Do everything the same as simple noisy channel discussed earlier, but
  - Calculate the edit distance over the pronunciation string instead
  - Then, select those with distance = 1 and continue

- Nowadays, a numerous number of distance metric exists,
  - Check — https://pypi.org/project/abydos/
  - One of the most highly recommended is **Jaro-Winkler**

Time for questions