SE 481 Introduction to Information Retrieval (IR for SE)

Module #4 — IR-based Spell Collections

College of Arts, Media and Technology
Chiang Mai University

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

- E.g.,

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

Rate of spelling errors

- Error rates fluctuate based on environment and application specifics.

- Typically ranges from 1% to 20%.

- Elevated on small keyboards, like those on smartphones.
- Increased within certain applications, such as web search queries.
- Amplified in languages with complex orthographies, like Thai with its vowels and tone marks.

Spelling error tasks

- Spelling error detection
  - Identifying incorrect spellings within text.

- Spelling error correction
  - **Auto-correct** — Automatically replace unrecognized words with a known alternative, often without user notification.
  - **Suggest a corrected word** — Offer a correction for unrecognized words upon detection.
  - **Suggest a list of possible corrected words** — Display a list of probable known words, ranked by the likelihood of accuracy.

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

- Words **absent** from a pre-established **dictionary** are flagged as errors.
- A more extensive dictionary generally yields better results, assuming computational resources are not a constraint.
- Dictionaries from noisy sources may lead to poor error detection.

- Approach
  - Create a list of potential correct words.
  - Then, score and rank these words to select the most contextually appropriate correction.

Real word & non-word spelling errors

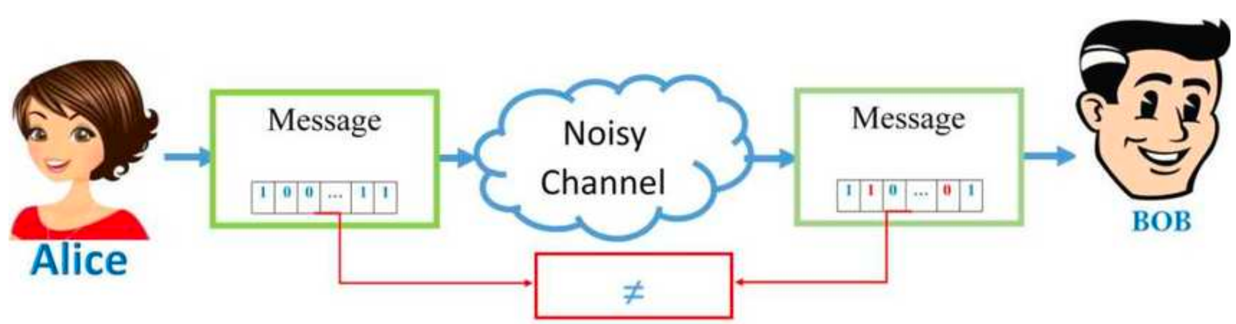
- For each word *w*, generate a set of candidates by:
  - Selecting words with similar pronunciations.
  - Choosing words with akin spellings.
  - Including *w* itself in the set.

- **Selecting the best candidate**
  - Apply the Noisy channel model to understand spelling errors.
  - Evaluate the context: check if surrounding words form a coherent sentence.
  - Example: 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 (k-grams)
  - e.g., good morning — \$g go oo od d\$ \$m mo or rn ni in ng g\$
  - \$ denotes a word boundary.

- Word bigrams (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 find the correct word w' for a misspelled 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

- Spelling similarity
  - Select words with **a minimal <u>edit distance</u>** from the misspelled word.

- Pronunciation proximity
  - Choose words with a close phonetic resemblance to the erroneous word.

Candidate testing

- Damerau-Levenshtein edit distance — Measures the minimum number of operations required to transform one string into another, with the following permissible edits:
  - Insertion of a character
  - Deletion of a character
  - Substitution of a character
  - Transposition of two adjacent characters

- 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	deet	-	f	Insertion
defet	deft	-	e	Insertion
defet	defer	r	t	substitution
defet	defeat	a	-	deletion
defet	defect	c	-	deletion

Candidate generation and error observations

- Spelling error insights
  - Approximately 80% of spelling errors occur within an edit distance of 1.
  - Nearly 100% of errors fall within an edit distance of 2.
  - Typos in the first letter of words are uncommon.

- Edit distance considerations
  - Insertions may involve adding spaces or hyphens to split a word incorrectly joined
    - e.g., **thismethod** to **this method**
  - Deletions may include removing spaces to correct words incorrectly separated
    - e.g., **data base** to **database**

Candidate generation — procedure

1. Edit distance calculation
    - Iterate through the dictionary, computing the edit distance between the query term and each dictionary entry.

2. Candidate shortlisting
    - Compile a list of dictionary words that are within an edit distance of 2 from the query term.

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)

CAMT 2023
College of Arts, Media and Technology
Chiang Mai University

Language model

- Probability estimation
  - Determine *P(w)*, the probability of a word *w*, using a representative corpus.

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

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

Example

- Utilizing the Corpus of Contemporary English (COCA — https://www.english-corpora.org/coca/) as the reference corpus.
- At of the end of 2019, T ~ 1,001,610,938 terms
- List of words was taken from https://www.dcode.fr/levenshtein-distance

Word	Frequency	*P(w)*	Rank
deet	420	0.000000419	5
deft	1,240	0.000001238	4
defer	2,237	0.000002233	3
defeat	21,940	0.000001240	1
defect	3,972	0.000003966	2

```
COCA = pd.DataFrame([['deet',420], ['deft',1240], ['defer', 2237], ['defeat',21940], ['defect',3972]], columns=['word','frequency'])
COCA_pop = 1001610938
COCA['P(w)'] = COCA['frequency']/COCA_pop
COCA['rank'] = COCA['frequency'].rank(ascending=False, method='min').astype(int)
```

Example

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

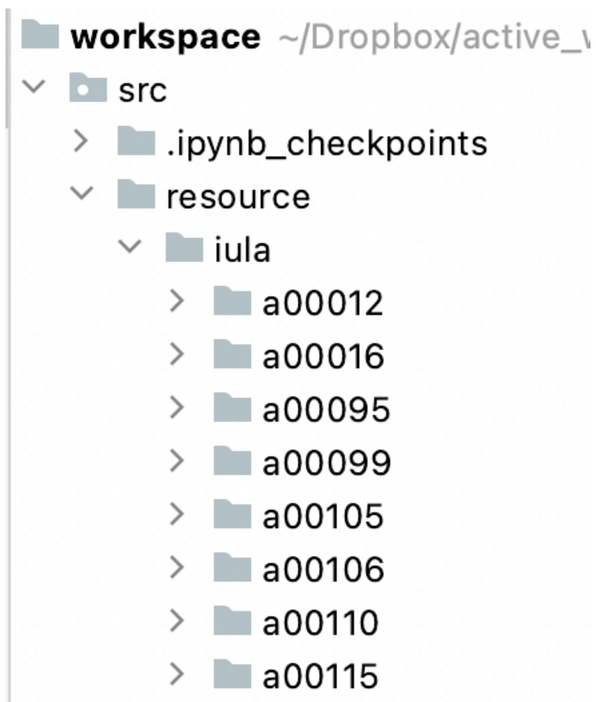
$$P(w) = \frac{C(w)}{T}$$

Word	Frequency	*P(w)*	Rank
deet	124	0.000000065	5
deft	814	0.000001238	3
defer	1,416	0.000000745	4
defeat	121,408	0.000001240	1
defect	7,793	0.000004102	2

```
WIKI = pd.DataFrame([['deet',124], ['deft',814], ['defer', 1416], ['defeat',121408],
['defect',7793]], columns=[‘word','frequency'])
WIKI_pop = 1.9e9
WIKI['P(w)'] = WIKI['frequency']/WIKI_pop
WIKI['rank'] = WIKI['frequency'].rank(ascending=False, method='min').astype(int)
```

Example — We can also build our own corpus

- IULA Spanish-English Technical Corpus
  (https://repositori.upf.edu/handle/10230/20052)
- Download, extract, move the sub directories under EN to resource folder



Example — If we have raw data

- IULA Spanish-English Technical Corpus (https://repositori.upf.edu/handle/10230/20052)

```python
topdir = 'resource/iula'
all_content = []
for dirpath, dirnames, filename in os.walk(topdir):
    for name in filename:
        if name.endswith('plain.txt'):
            with open(os.path.join(dirpath, name)) as f:
                all_content.append(f.read())

processed_content = [m3.preProcess(s) for s in all_content]
```

```python
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(processed_content)
freq_iula = vectorizer.transform(processed_content)
freq_iula = pd.DataFrame(freq_iula.todense(), columns=vectorizer.get_feature_names_out()).sum()
```

Example — If we have raw data

- IULA Spanish-English Technical Corpus (https://repositori.upf.edu/handle/10230/20052)

```
query = ['deet', 'deft', 'defer', 'defect', 'defeat']
transformed_query = [vectorizer.inverse_transform(vectorizer.transform([q])) for q in query]
query_freq = pd.Series([freq_iula.T.loc[tq[0]].values[0] if len(tq[0]) > 0 else 0 for tq in transformed_query], index= query)
```

```
IULA = pd.DataFrame(query_freq, columns=['frequency'])
IULA_pop = total
IULA['P(w)'] = IULA['frequency']/IULA_pop
IULA['rank'] = IULA['frequency'].rank(ascending=False).astype(int)
```

Example

- Utilizing to IULA Spanish-English Technical Corpus (https://repositori.upf.edu/handle/10230/20052)
- T ~ 169,176

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

Word	Frequency	*P(w)*	Rank
deet	0	0.000000000	4
deft	0	0.000000000	4
defer	5	0.000029555	3
defeat	8	0.000047288	2
defect	72	0.000425592	1

Corpus does matter

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)

<table>
<tr><th>Candidate correction</th><th>Correct letter</th><th>Error letter</th><th>x|w</th><th colspan="2">P(x|w)</th></tr>
<tr><td>deet</td><td>-</td><td>f</td><td>f |</td><td>0 / 120,870</td><td>0.000000</td></tr>
<tr><td>deft</td><td>-</td><td>e</td><td>e |</td><td>2 / 632,999</td><td>0.000003</td></tr>
<tr><td>defer</td><td>r</td><td>t</td><td>t | r</td><td>11 / 309,545</td><td>0.000036</td></tr>
<tr><td>defeat</td><td>a</td><td>-</td><td>e | ea</td><td>354 / 27,583</td><td>0.012833</td></tr>
<tr><td>defect</td><td>c</td><td>-</td><td>e | ec</td><td>47 / 14,841</td><td>0.003167</td></tr>
</table>

P(x|w)

```python
def get_count(c,norvig_orig):
    return norvig_orig.apply(lambda x: x.term.count(c) * x.freq, axis=1).sum()
```

```python
character_set = list(map(''.join, itertools.product(ascii_lowercase, repeat=1)))
+ list(map(''.join, itertools.product(ascii_lowercase, repeat=2)))
```

```python
pool = Pool(8) #8 is your #compute cores
    freq_list = pool.starmap(get_count, zip(character_set, itertools.repeat(norvig_orig)))
```

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

```python
COCA['P(x|w)'] = [
        (0 / freq_df.loc['f'].values)[0],   #deet
        (norvig.loc['e| '].values / freq_df.loc['e'].values)[0],   #deft
        (norvig.loc['t|r'].values / freq_df.loc['r'].values)[0],   #defer
        (norvig.loc['e|ea'].values / freq_df.loc['ea'].values)[0], #defeat
        (norvig.loc['e|ec'].values / freq_df.loc['ec'].values)[0]  #defect
]
```

$P(x|w)P(w)$ — Using COCA

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

Candidate correction	frequency	P(w)	rank	P(x|w)	10^9 * P(x|w)P(w)
deet	420	0.000000419	5	0.000000	0.000000000
deft	1,240	0.000001238	4	0.000003	0.003911556
defer	2,237	0.000002233	3	0.000036	0.079366242
defeat	21,940	0.000001240	1	0.012833	281.124908619
defect	3,972	0.000003966	2	0.003167	12.558705434

$P(x|w)P(w)$ — Using IULA

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

Candidate correction	frequency	P(w)	rank	P(x|w)	10^9 * P(x|w)P(w)
deet	0	0.000000000	4	0.000000000	0.000000000
deft	0	0.000000000	4	0.000003160	0.000000000
defer	5	0.000029555	3	0.000035536	1.050268025
defect	72	0.000425592	1	0.003166902	1,347.809263654
defeat	8	0.000047288	2	0.012833992	606.894214383

Estimating spelling corrections

- **Corpus data**
  - Provides an estimated probability for the appearance of each word.
- **Misspelt statistics**
  - Offers estimated frequencies for common misspellings.
- **Correction likelihood**
  - The product of the corpus probability and misspelling frequency helps determine the most probable correct word for a given misspelling.

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, e.g.,

- 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.

- Research indicates that 25% to 40% of spelling errors result in real words.
- This highlights the importance of context in detecting and correcting real-word errors, as standard spellchecks may not flag them.

Approach

- Candidate generation for each word
  - Create a set of possible corrections that includes:
    - The word itself, assuming it may be correct.
    - All dictionary words within an edit distance of 1.
    - Homophones, or words that sound the same but are spelled differently.
- Selection of best candidates
  - Employ a modified version of the Noisy Channel Model to select the most probable correct word from the candidate set.

Noisy channel for real-word spell correction

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

$$\text{Candidate}(x_1) = \{x_1, w_1, w'_1, w''_1, ...\}$$
$$\text{Candidate}(x_2) = \{x_2, w_2, w'_2, w''_2, ...\}$$
$$\text{Candidate}(x_3) = \{x_3, w_3, w'_3, w''_3, ...\}$$

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

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

Incorporating context

- When lacking a specialized corpus, discerning between similar words like **defeat** or **defect** necessitates analysis of the surrounding context.

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

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

  - Calculated by the frequency of word pairs divided by the frequency of the preceding word.

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(a)	607 / 21,888,145	0.0000277
P(defect | a)	C(a defect) / C(a)	453 / 21,888,145	0.0000206
P(free | defeat)	C(defeat free) / C(defeat)	1 / 21,940	0.0000455
P(free | defect)	C(defect free) / C(defect)	5 / 3,972	0.0012588

- *P(“a defeat free”)* = 0.0000277 x 0.0000455 = 0.00000000126
- P(“a defect free”) = 0.0000206 x 0.0012588 = 0.00000002593 ←

Incorporating context

- The impact of choosing different corpora diminishes when contextual information is incorporated into the analysis.

Improved the edit distance component

- Pronunciation as a unit
  - Extend the basic unit of edit distance from characters to phonemes, considering how words sound.

- Assumption of a noise-free channel
  - Implement a distance function that captures phonetic similarity to identify the closest sounding word in the corpus.

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

- Adaptation for pronunciation
  - Follow the same process as the standard noisy channel, but calculate edit distances using phonetic representations of words.
- Selection criterion
  - Prioritize candidates with a phonetic edit distance of 1 for further evaluation.
- Exploring Distance Metrics
  - Explore different metrics for better accuracy in matching pronunciations.
    — https://pypi.org/project/abydos/
  - Consider using the **Jaro-Winkler** distance for its effectiveness in comparing string similarity.

Assignment

- Using the approach discussed in this chapter and show the ranking of spelling correction candidates of **a mistyped word with at least 4 candidates**.
  - All the candidates are of **1 edit distance** away from your selected word.
  - Try using **IULA** first since you are almost able to automate the entire process.

Time for questions