Spelling Correction

(Search and Data Mining)

Hui Fang
Department of Electrical and Computer Engineering
University of Delaware

Reference

- How to write a spelling corrector, by Peter Norvig

http://norvig.com/spell-correct.html

Types of Spelling Errors

- Errors are not real words.
  - *elephent* →*elephant*
- Errors are real words.
  - *three* → *there*
  - *their* → *there*

How to Detect and Correct Non-word Spelling Errors

- Check whether the word is in a dictionary.
- If no, need to generate correction
  - Generate candidates
  - Choose the one which is best

Candidate Generation

- Find real-words that are similar to the errors
  - Similar spelling
  - Similar pronunciation
- Use minimal edit distance between two strings to measure the similarity
  - Edits include insertion, deletion, substitution and transposition of two adjacent letters.
  - 80% of errors are within edit distance 1
  - Almost all errors within edit distance 2

Ranking Candidates

- Given a misspelled word x, we can rank possible correct words w based on

$$\hat{w} = \underset{w \in candidates}{\operatorname{argmax}} P(w \mid x)$$
$$= \underset{w \in candidates}{\operatorname{argmax}} \frac{P(x \mid w)P(w)}{P(x)}$$
$$= \underset{w \in candidates}{\operatorname{argmax}} P(x \mid w)P(w)$$

Ranking Candidates

- Given a misspelled word x, we can rank possible correct words w based on

$$\hat{w} = \underset{w \in candidates}{\text{argmax}}\ P(w \mid x)$$

$$= \underset{w \in candidates}{\text{argmax}}\ \frac{P(x \mid w)P(w)}{P(x)}$$

$$= \underset{w \in candidates}{\text{argmax}}\ P(x \mid w)P(w)$$

Go through all candidates, and choose the one that gives the best combined probability score

Ranking Candidates

- Given a misspelled word x, we can rank possible correct words w based on

$$\hat{w} = \underset{w \in candidates}{\text{argmax}}\ P(w \mid x)$$

$$= \underset{w \in candidates}{\text{argmax}}\ P(x \mid w)P(w)$$

How likely do you see w in a general collection?

Can be estimated using any language models, such as unigram models.

Ranking Candidates

- Given a misspelled word x, we can rank possible correct words w based on

$$\hat{w} = \underset{w \in V}{\text{argmax}}\ P(w \mid x)$$

$$= \underset{w \in V}{\text{argmax}}\ \frac{P(x \mid w)P(w)}{P(x)}$$

$$= \underset{w \in V}{\text{argmax}}\ P(x \mid w)P(w)$$

You want to type x, how likely is it to type x instead?

Error Model : P(x|w)

- $P(x|w)$ = probability of the edit
  - deletion[x,y]: count(xy typed as x)
  - insertion[x,y]: count(x typed as xy)
  - substituion[x,y]: count(x typed as y)
  - transposition[x,y]: count(xy typed as yx)

It requires lots of training data!

A simplified error model : P(x|w)

- A simplified solution without training data set
  - All known words of edit distance 1 are infinitely more probable than known words of edit distance 2.
  - All known words of edit distance 1 are infinitely less probable than a known word of edit distance 0.

Reference: How to write a spelling corrector, by Peter Norvig, http://norvig.com/spell-correct.html

Peter Norvig's Spelling Corrector

- Not industrial-strength spell corrector
- A toy non-word spelling corrector
  - 80 or 90% accuracy
  - At a processing speed of at least 10 words per second

Reference: How to write a spelling corrector, by Peter Norvig, http://norvig.com/spell-correct.html

```
import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

Reference: How to write a spelling corrector, by Peter Norvig, http://norvig.com/spell-correct.html

15

Recall: Candidate Generation

- Find real-words that are similar to the errors
  - Similar spelling
  - Similar pronunciation
- Use Edit distance to measure the similarity
  - 80% of errors are within edit distance 1
  - Almost all errors within edit distance 2

16


```
import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word))
    return max(candidates, key=NWORDS.get)
```

Returns a set of all words w that are one edit away from x

Consider words that are two edits away.

Reference: How to write a spelling corrector, by Peter Norvig, http://norvig.com/spell-correct.html

17

Recall: Ranking Candidates

- Given a misspelled word x, we can rank possible correct words w based on

$$\hat{w} = \underset{w \in candidates}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in candidates}{\operatorname{argmax}} P(x \mid w) P(w)$$

How likely do you see w in a general collection?

Can be estimated using any language models, such as unigram models.

18


```
import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

NWORDS[w] holds a count of how many times that word w has been seen in the training file. (add one smoothing)

Reference: How to write a spelling corrector, by Peter Norvig, http://norvig.com/spell-correct.html

19

Recall: Ranking Candidates

- Given a misspelled word x, we can rank possible correct words w based on

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w) P(w)}{P(x)}$$

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w) P(w)$$

You want to type x, how likely is it to type x instead?

20

```
import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lam
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').

alphabet = 'abcdefghijklmnopqrstuvwxy

def edits1(word):
   splits     = [(word[:i], word[i:])
   deletes    = [a + b[1:] for a, b i
   transposes = [a + b[1] + b[0] + b[
   replaces   = [a + c + b[1:] for a,
   inserts    = [a + c + b     for a, b
   return set(deletes + transposes + replac

def known_edits2(word):
    return set(e2 for e1 in edits1(word for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
```

Without training set, defined a trivial model: all known words of edit distance 1 are infinitely more probable than known words of edit distance 2, and infinitely less probable than a known word of edit distance 0.

Reference: How to write a spelling corrector, by Peter Norvig, http://norvig.com/spell-correct.html

UNIVERSITY of DELAWARE

UNIVERSITY of DELAWARE

Solving real-word spelling errors

- Example:
  - It takes her about 6 *minuets* to go back home.
- Methods:
  - For each word in sentence
    - Generate *candidate set*
      - the word itself
      - all single-letter edits that are English words
      - words that have the same pronunciation
  - Choose the most likely combination

23