MTSpell

improved spelling correction
for post-editing and interactive MT

{Marco, Chara, Uli, Herve, Christian}

More resources ⇨ better suggestions

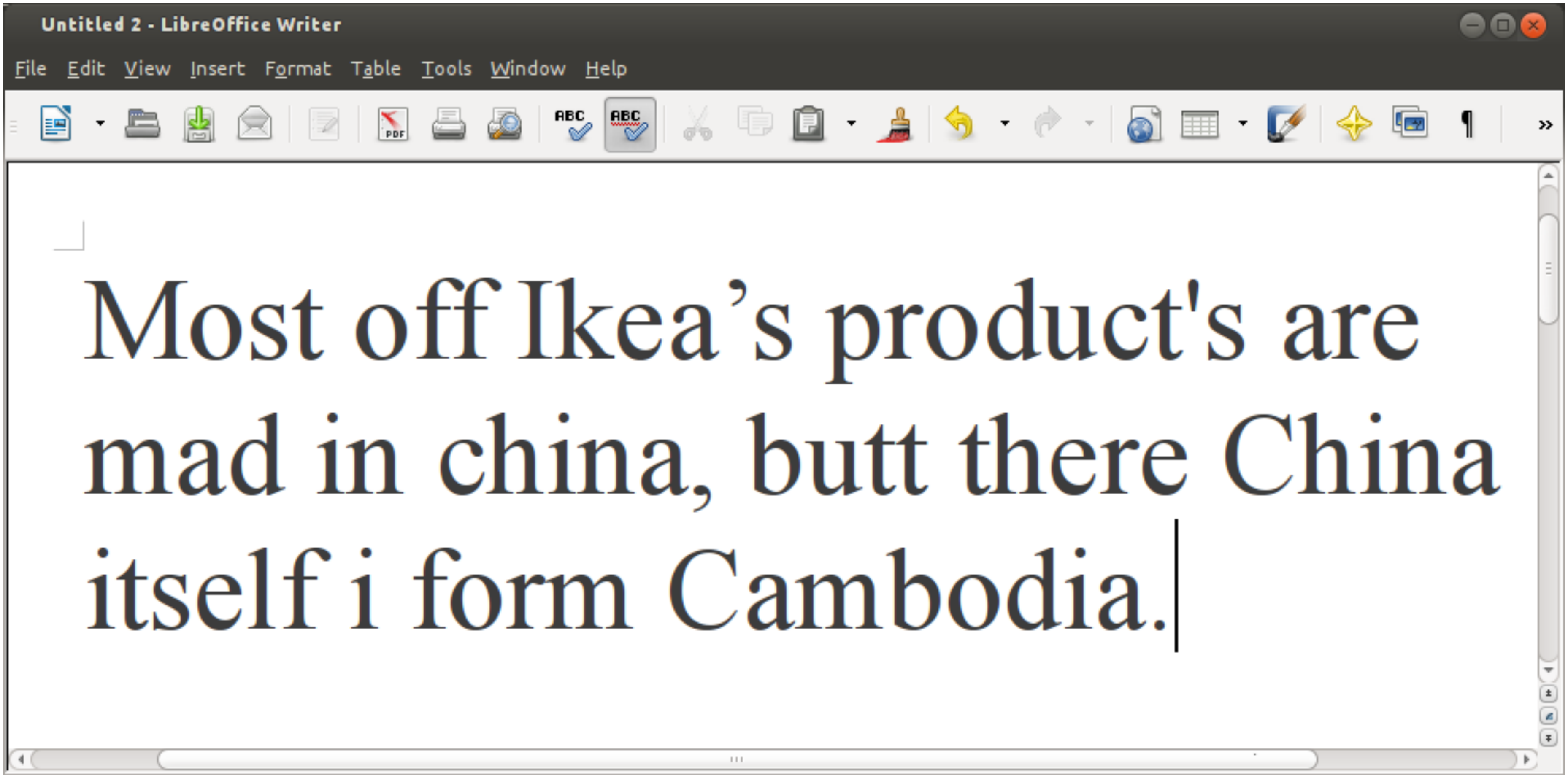
Bad:

- Common spell checkers (aspell and friends) limited to single word
- Best at suggesting common words

Better:

- Look at the context
- Use more RAM
- Use the source, Luke

Real-word errors



Pipeline

1. Find correction candidates
   a. Edit distance
   b. Split words
   c. Join words
2. Score locally
3. Produce search graph
4. Score with LM
5. Cross fingers

Levenshtein Distance

- The minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other

- e.g. Lev(from, form) = 2
  ['A', 'D', 'A', 'I', 'A']
  - where:
    - A: aligned = 3 (count 0)
    - D: deleted = 1 (count 1)
    - I: Inserted = 1 (count 1)

Levenshtein Distance

- More sensitive measure (feature)

- Two Variations:
  a. different weights for each edit based on the **letters involved**
     - high probability to misspell letter ‘s’ with letter ‘z’

Levenshtein Distance

- More sensitive measure (feature)

- Two Variations:
  a. different weights for each edit based on the letters involved
     - high probability to misspell letter ‘s’ with letter ‘z’

  b. different weights for each edit based on the **edit position in the words**
     - high probability to adjust morphology at the end of the word

Letter-Weighted Lev. Distance

- Weight differently edits according to the letters that are involved
  - *‘s’* into ‘*z’* more probable than ‘*s’* into ‘*k’*

- Given an annotated corpus,
  - compute the substitution matrix:
    - count how often letter ‘*j’* in the misspelled word is replaced by ‘*i’* in the correct word
    - for each letter pair, compute the probability of replacing ‘*j’* with ‘*i’*
  - in testing, use the probability as weight of each edit

Letter-Weighted Lev. Distance

- Toy example:

Lev(from, *) = 2

wLev(from, frim) = 0.985

wLev(from, frlm) = 0.992

wLev(from, fram) = 0.995

wLev(from, frxm) = 1

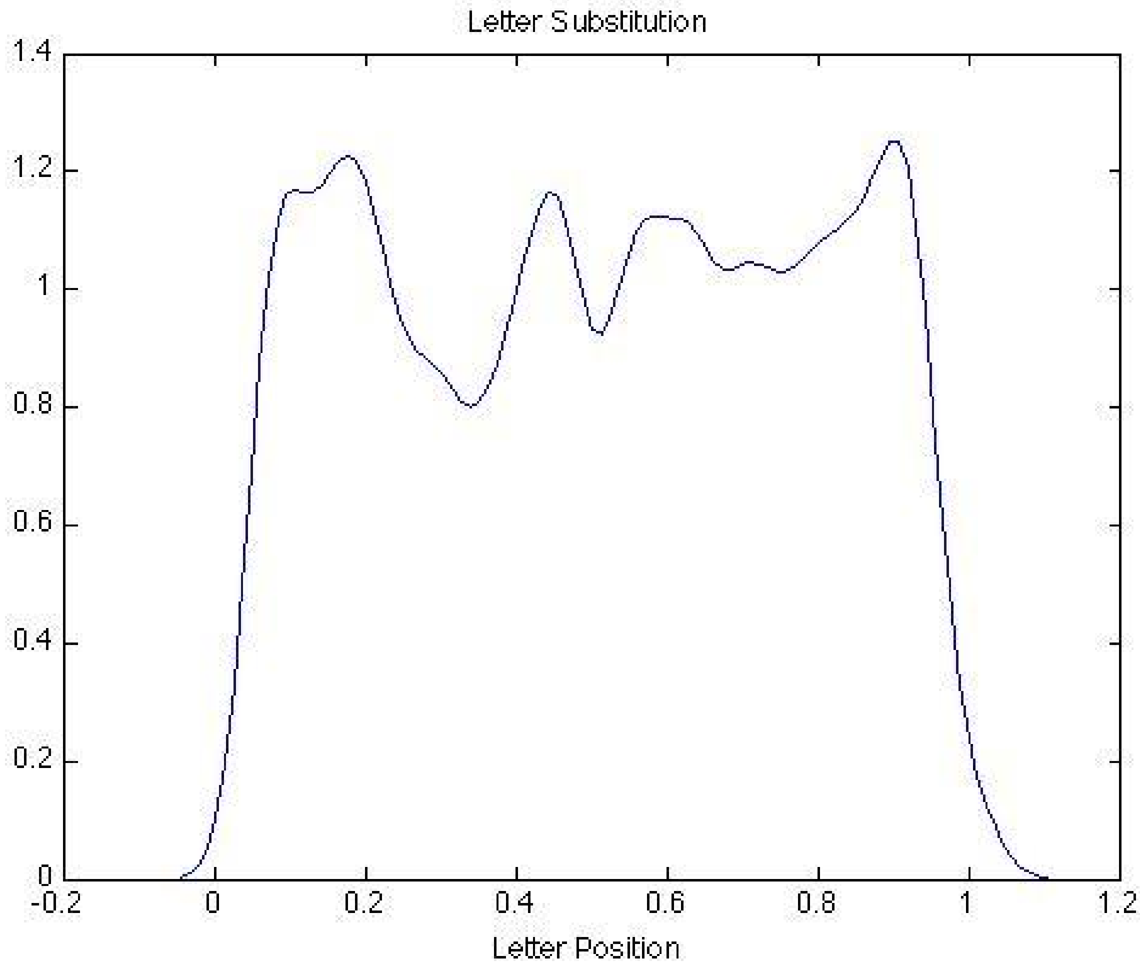
Position-Weighted Lev. Distance

- Weight edits differently according to their positions in the words
  - corrections at the end of the word are more probable than at the beginning

- Given an annotated corpus:
  - count how often an error appears in a certain position
  - smooth the counts using the kernel density estimation
  - in testing, use this probability as weight of each edit

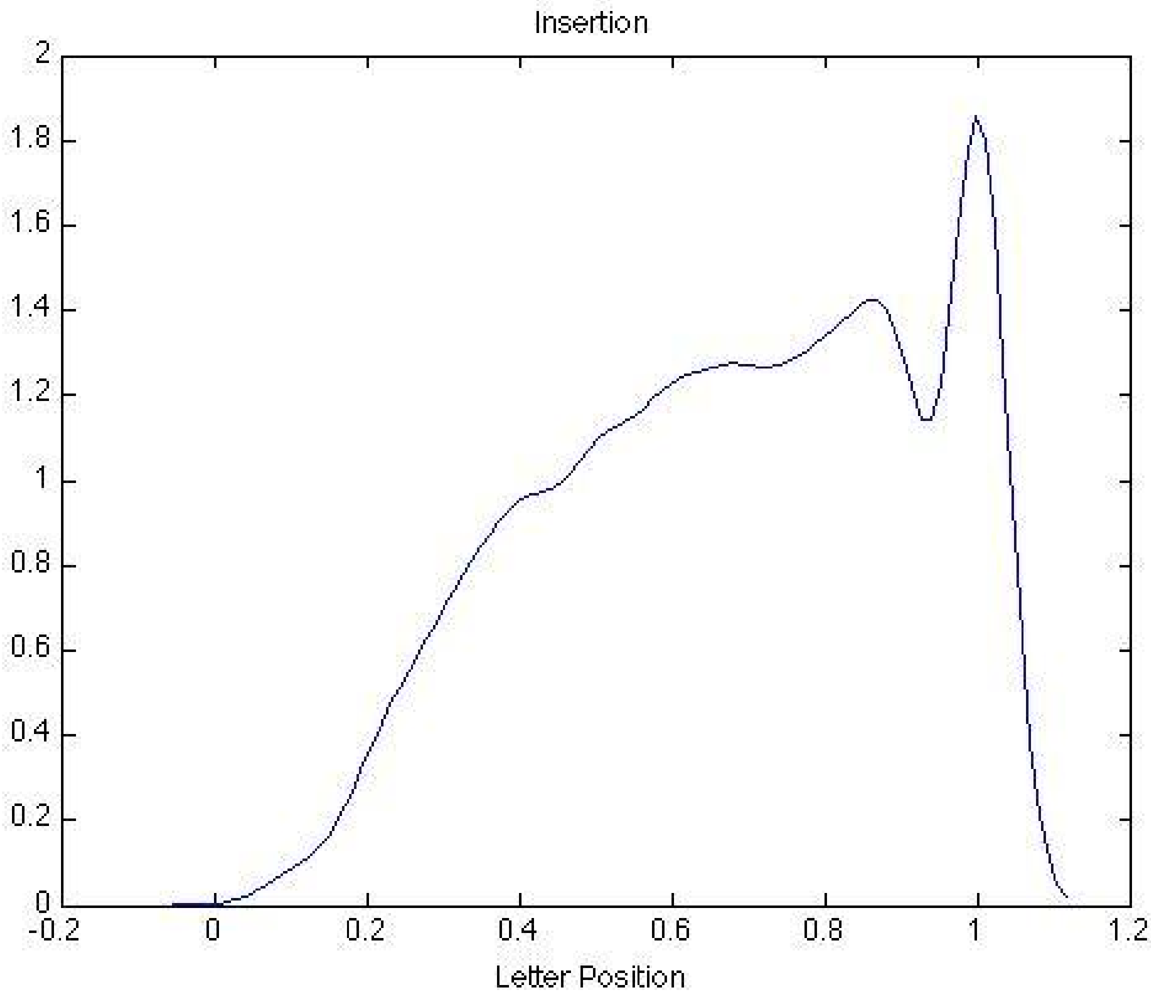
Position-Weighted Lev. Distance

Substitution:



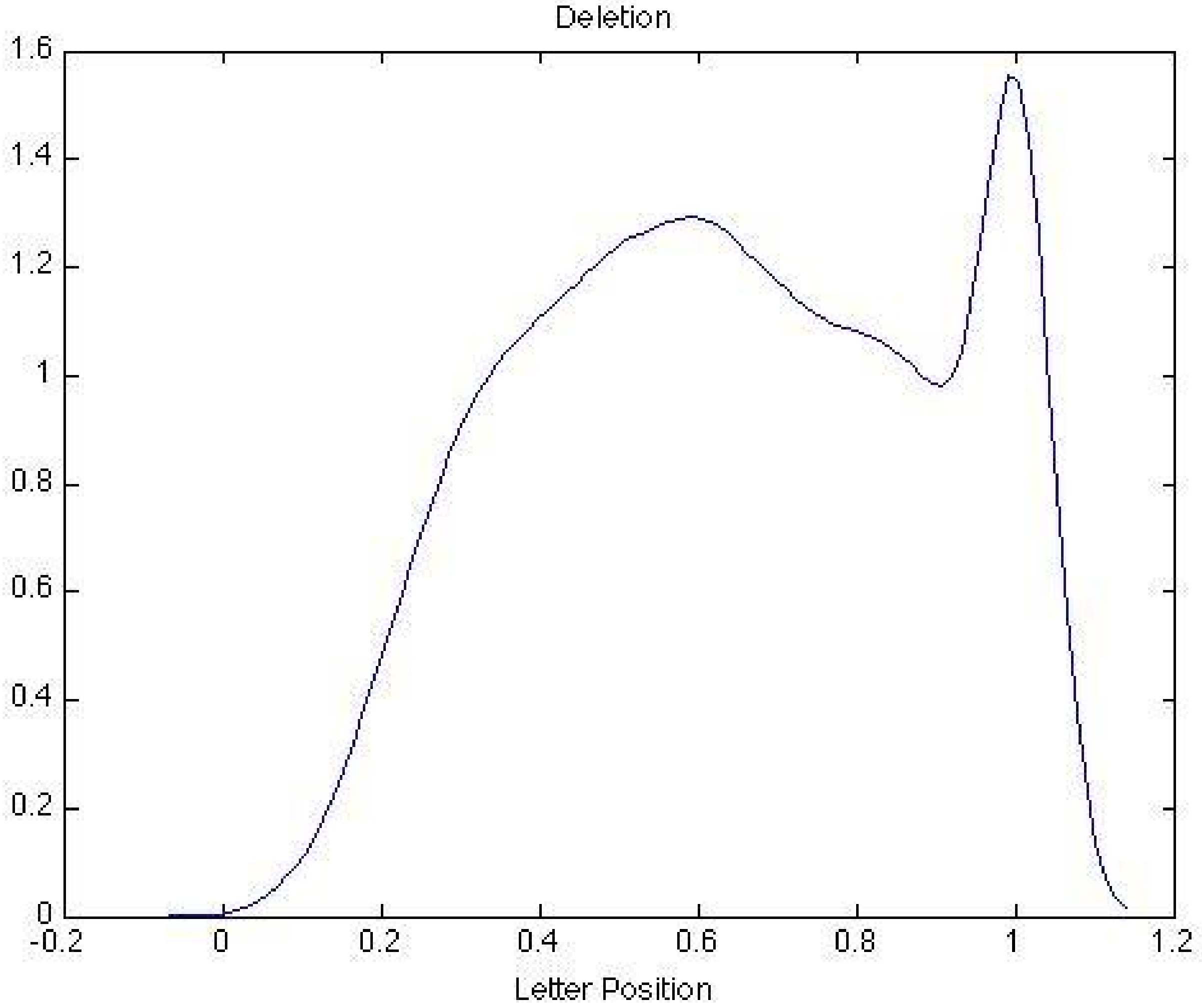
Position-Weighted Lev. Distance

Insertion:



Position-Weighted Lev. Distance

Deletion:



Position Weighted Lev. Distance

- Toy example:

Lev(from, *) = 2

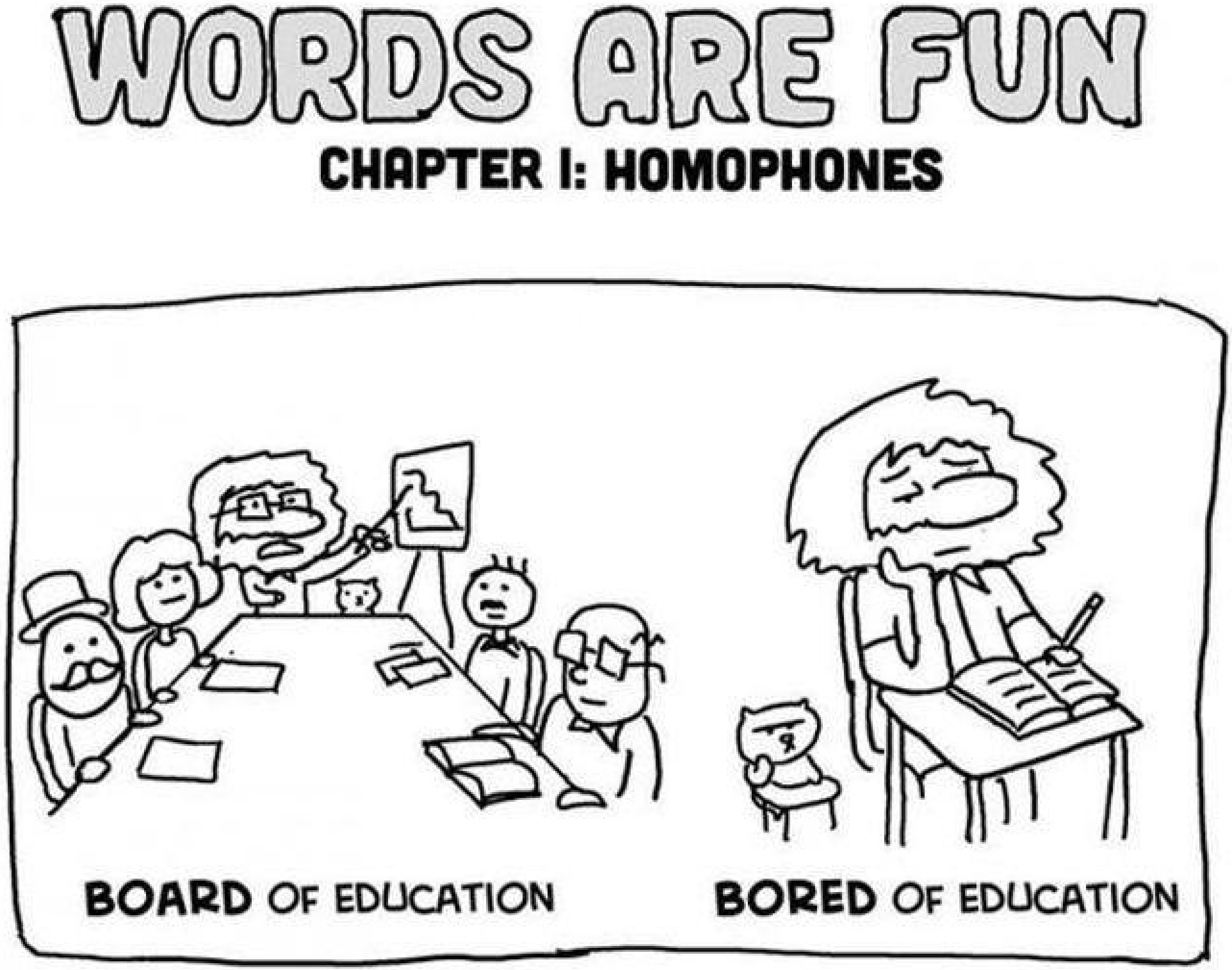
pwLev(from, irom) = 0.106

pwLev(from, fiom) = 0.799

pwLev(from, frim) = 1.047

pwLev(from, froi) = 0.238

Phonetic algorithm



Phonetic algorithm

Homophones may have EditDistance > 1

- Soundex algorithm:

  e.g. czech: C200
  check: C200

  faster than other phonetic algorithms (e.g. NYSIIS, Double Metaphone)

Finding correction candidates

Naive approach:

- for each item in the dictionary, compute edit distance to word in question

Peter Norvig's algorithm

- systematically distort word in question by inserting, deleting, transposing etc. letters and checking if they are in the dictionary (http://norvig.com/spell-correct.html)

Finding correction candidates

Faroo algorithm (100,000 times faster for ed=3)

- for each word in the dictionary, systematically remove up to *n* letters
- build a map from each of the resulting strings to the original string
- at lookup time, delete up to *n* words from the word in question, consult the map from step 2
- compute edit distance for each candidate word found this way

(http://blog.faroo.com/2012/06/07/improved-edit-distance-based-spelling-correction/)

Done so far

- Naive approach in Python (works)
- Faroo algorithm in C++ with MPH for indexing (also works, yay!)

Finding correction candidates

Split words

Not just simple segmentation:

haveto ⇨ have to
mydag ⇨ ?

renew list of candidates for misspelled word

Finding correction candidates

Split words

```
for all possible splits:
    for left split in dictionary(edit distance <=1):
        for right split in dictionary(edit distance <=1):
            add to candidates
```

Finding correction candidates

Split words

```
for all possible splits:
    for left split in dictionary(edit distance <=1):
        for right split in dictionary(edit distance <=1):
            add to candidates
```

e.g. m-ydag **my-dag** myd-ag myda-g

(my day, my dog)

Progress

Progress

- Candidates
- Scores
- FAST candidate

Ongoing:

- Splits / Joins

Soon

- Evaluation

Example 1

```
$ echo "Kissed a girl one night and here iyes
were burning blue" | ./spell.py -mincount=1000 -
dist 2 -counts dict/english.counts > data/0
read 61036 entries from dict/english.counts with
min count 1000
```

```
$ decode -i data/ -l 10M.kenlm -K 1000 --weight
WordPenalty=0 LanguageModel=1.0
LanguageModel_OOV=-10 EditDistance=-2
SoundMap=1 WeightedEditDistance=-10
0 |||  kissed a girl one night and her eyes were
burning blue
```

Example 2

```
$ echo "they hade cleand the river and made it very wide fore the ducks" | ./spell.py -mincount=1000 -dist 2 -counts dict/english.counts > data/2
```

```
$ decode -i data/ -l 10M.kenlm -K 1000 --weight WordPenalty=0 LanguageModel=1.0 LanguageModel_OOV=-10 EditDistance=-2 SoundMap=1 WeightedEditDistance=-10
0 ||| they have cleaned the river and made it very wide for the ducks
```