An abstract description of peter Norving algorithm

Our training dataset is a text file called **big.txt** which has a large number or correct-spelling words.

The test set is a text file called **spell-testset.txt** that has a correct word in each line, a colon and some examples of incorrect-spelling words for the same word.

An abstract description of how the code is running and functions work

-> First of all, the funciton **P():** It returns the probability of the word,

how it works: it devides number of word repeats within the document / number of words in the whole document.

-> The function **Correction()** : It returns the most probable spelling correction for the word

how it works: it takes a word as a parameter and returns the result of the called function **Max()** which return the maximum probability word of possible words.

-> The function **Candidates()**: It generates possible spelling corrections for a word

how it works: It calls the function **Known()** with different parameters contain possible words

known([word]): Calls it by the argumented word

known(edits1(word)): Calls it by the resulted set from the function **edit1()** with the same word.

known(edits2(word)): Calls it by the resulted set from the function **edit2()** with the same word

-> The function **Known()**: It takes a word as a parameter and returns the subset of words that appear in the dictionary of WORDS

how it works: It checks if the word which is sent as a parameter exists in the WORDS doc, it returns the same word. But if the word doesn't exist in the WORDS doc, it returns an empty set.

-> The function **edit1()**: It takes a word as a parameter and returns all edits that are one edit away from the word.

how it works: It returns a set of many words that is made by several ways to get ALL edits of the same word. For an example the transposes matrix, it transposes the first 2 letters, the second 2 letters and so on.

-> The function **edit2()**: *All edits that are two edits away from `word` using* **edit1()**
how it works: It just calls the function **edit1()** with the word which is made up as a result of the function **edit2()** before.

The function **SpellTest()**: "Run correction(wrong) on all (right, wrong) pairs; report results. Then calculate accuracy "

- **Classic NLP Approach**

Norvig's Spelling Corrector

The idea is if we artificially generate all terms within maximum edit distance from the misspelled term, then the correct term must be among them. We have to look all of them up in the dictionary until we have a match. So all possible combinations of the 4 spelling error types (insert, delete, replace and adjacent switch) are generated. This is quite expensive with e.g. 114,324 candidate term generated for a word of length=9 and edit distance=2.

SymSpell Algorithm

SymsSpell is an algorithm to find all strings within a maximum edit distance from a huge list of strings in very short time. It can be used for spelling correction. SymSpell derives its speed from the Symmetric Delete spelling correction algorithm and keeps its memory requirement in check by prefix indexing.

The Symmetric Delete spelling correction algorithm reduces the complexity of edit candidate generation and dictionary lookup for a given Damerau-Levenshtein distance. It is six orders of magnitude faster (than the standard approach with deletes + transposes + replaces + inserts) and language independent.

- **Machine Learning Approach**

Word2Vec

It is an adaptation of Peter Norvig's spell checker. It uses word2vec ordering of words to approximate word probabilities. Indeed, Google word2vec apparently orders words in decreasing order of frequency in the training corpus. This kernel requires to download Google's word2vec: https://github.com/mmihaltz/word2vec-GoogleNews-vectors
mmihaltz/word2vec-GoogleNews-vectors
word2vec Google News model . Contribute to mmihaltz/word2vec-GoogleNews-vectors