SpellChecker

SpellChecker

A Spell Checker allows to suggest a list of words similar to a misspelled word. This implementation is based on David Spencer's code using the n-gram method and the Levenshtein distance.

Structure of a dictionary index

An index (the dictionary) with all the possible words (a lucene index) must be created. The structure of this index is (for a 3-4 gram) this:

Index Structure	Example
word	kings
gram3	kin, ing, ngs
gram4	king, ings
start3	kin
start4	king
end3	ngs
end4	ings

Import: Adding Words to the Dictionary

We can add the words coming from a Lucene Index (more precisely from a set of Lucene fields), and from a text file with a list of words.

- Example: we can add all the keywords of a given Lucene field of my index.

```
SpellChecker spell= new SpellChecker(dictionaryDirectory);
spell.indexDictionary(new LuceneDictionary(my_luceneReader,my_fieldname));
```

Getting a List of Suggested Words

The suggestSimilar method returns a list of suggested words sorted by:

1. the Levenshtein distance (the most similar word to the misspelled word is the first in the list). 2. (optionally) the popularity of the word in a given Lucene Field.

Furthermore, that list can be restricted only to the words present in a given Lucene Field.

- First example: the suggestSimilar(misspelled_word, num_list) method.
  The *num_list* is the maximum number of words returned.
  In this example the list is just sorted with the Levenshtein distance.

```
String[] l=spellChecker.suggestSimilar("sevanty", 2);
//l[0] = "seventy"
```

- Second example: the suggestSimilar(misspelled_word, num_list, myIndexReader,myField, morePopular) **Note**: if myIndexReader and myField are null this method is the same as the first method
  1. The returned words are restricted only to the words presents in the field *myField* of the Lucene Index "myIndexReader" 2. The list is also sorted with a second criterium: the popularity (the frequency) of the word in the user field 3. If *morePopular* is true and the mispelled word exists in the user field, return only the words more frequent than this.
     See the test case code for an example.

Changes

Version 1.1 :

- sort fixed (the sort was inversed!)

- set gram dynamically (depending of the length of the word)
- use the FuzzyQuery score: ((edit distance)/(length of word))
- new Dictionary interface + LuceneDictionary and PlaintextDictionary implementation
- replace addWords method by indexDictionary(Dictionnary dic)
- add a new public method: boolean exist(word)
- add a build.xml

Credits

- Maisonneuve Nicolas
- Spencer David