April 17 2015

Computer Science 51

TF: Allison Buchholtz-Au

Limor Gultchin, gultchin@college.harvard.edu
Meg Knister, mknister@college.harvard.edu
Zac Bathen, zbathen@college.harvard.edu
Gal Koplewitz, galkoplewitz@college.harvard.edu

Final Project Specification – Autocorrect

Brief Overview and Features

We hope to implement an effective Autocorrect application, which operates on some text as it is being written, and automatically replaces misspelled words with more likely alternatives. This will require various "moving parts" specified in the 'Featured' section below -- but it basically boils down to:

(1) An effective data structure for the dictionary for storing words, that can be accessed quickly enough for real time comparisons.

(2) A word-similarity algorithm, that ranks different possibilities from the dictionary based on evaluation of # of same letters, ordering, etc. This is an 'upgraded spellcheck,' in the sense that it indicates not only if spelling is wrong, but also offers corrections.

For the above two, we will be using algorithms described in the following resources (online guides, as well as academic articles on language processing):

http://norvig.com/spell-correct.html
http://www.lleess.com/2013/03/auto-correct-algorithm-in-python.html
http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/36180.pdf
http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=52A3B869596656C9DA285DCE83A0339F?doi=10.1.1.146.4390&rep=rep1&type=pdf

(3) Running corrections in real time -- this will require some form of event handling. The more feasible version is after every word (so run after space is pressed), and more difficult version could evaluate and offer options after every letter is clicked. Should not be too difficult to find online resource for doing this in Python.

(4) User interface -- whether textbox, web-based, etc.

Cool Additions

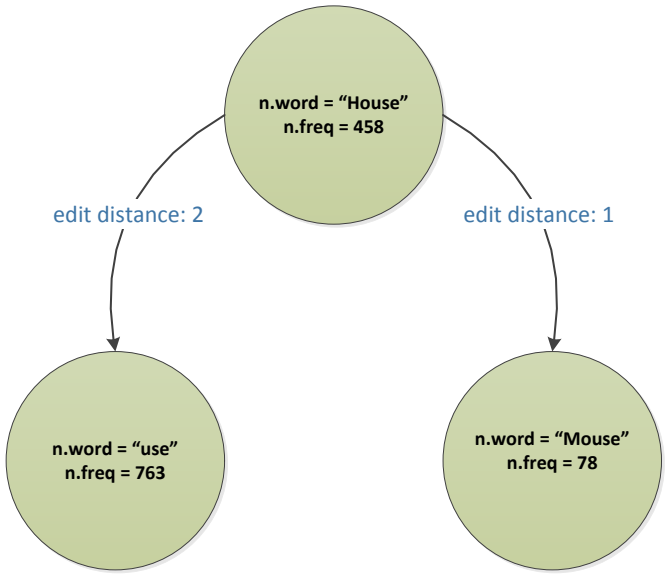
(5) Basic Machine learning for expanding dictionary with new words used frequently by user. This could be done in various ways (physical addition by user, used over a certain number of times, etc).

(6) Using different, topical dictionaries that offer corrections in line with a theme (e.g. Pirate dictionary, that corrects "friend" to "matey", "car" to "ship", "man" to "sea rat", "bird" to "parrot", etc).

Technical Specification

- **Dictionary module – BK Trees**

  We will be using BK trees as the preferred datatype for storing the dictionary. We used the guide found here. Our "nodes" will basically include 2 attributes – n.word, a string of the word, and n.freq, an integer value of the # of times this word appears in written English (based on parsing of common texts, see below). Illustration of BK-Tree:



The Methods in Bk-Trees

1. bk.add("word") – takes in a single string, and adds to the dictionary. If the string already exists, then increments the value of n.freq by 1. Edge weights (corresponding to edit distance), places to which nodes are added in the tree, are based on fundamentals of BK-trees. Returns unit (changes mutable data-structure). Note – this makes use of a helper function, that takes in two words, and calculates the edit distance between them (say bk.edit_dist("word1","word2")).
2. bk.add_text(text.txt) – takes in a txt file, parses it (changes to lowercase, handles various punctuation marks), and then calls bk.add on all the words in the resulting parsed text. We will run this on several large books, in order to reach a reasonable frequency estimate. Returns unit (changes mutable data-structure).
3. bk.search("word",int) – looks up possible words and words matches in the dictionary. This returns a list of tuples of matches within "int" **edit distances** away. For example, if we searched for ("thew",1), it might return: [("the",234);("thaw",123)] (the second element in the tuple is the frequency).
   If the word exists (there is a match for running on ("word",0)), the replace function will just ignore it (or replace, doesn't really matter).
4. bk.delete function – optional, but currently don't think we'll be deleting from our trees.

- **Correct Module**

  This part is the "heart" of the autocorrect algorithm. Basically, our function will be **correct("word")** – it will take in a word, and either return it the way it is (if in dictionary), or return a replacement (see display module) with the most likely alternative. First, it will run bk.search on it, returning a list of tuples, ordered by frequency (the second element in the tuple). Then, it analyses that list based on a series of factors – frequency, keyboard proximity, whether it is 1 or 2 edits away, etc – and returns the word that "scores" highest based on these, to replace the misspelled one.

- **Display + Event Module**

In a sense, this is our "main". We will use either a terminal based interface or something more complicated (see here: https://code.google.com/p/npyscreen/). First, we will initialize our BK-Tree, calling bk.add_text() on a bunch of texts, filling it with nodes of words+frequencies.

Then, we basically have a text window open, in which the user can type freely, and which displays user input in real time. We will have an event listener in place, which call the handler function every time the user hits space (stretch goal – every time a new letter is typed). The Handler then first calls correct("word"), "word" being the string that ended with the space bar, and then calls replace() with the string that correct returned.

Roadmap

Week 1

- Complete Codeacademy Python tutorial
- Read academic articles on correctness methods (based on edit distance, keyboard proximity, etc).
- Design the algorithm to actually do this ranking (pseudocode).
- Research best display/console based text interfaces.

Week 2

- Code up BK-tree.
- Implement correct().
- Write display and event module.
- If there is time, do stretch goals: Machine learning, fun dictionaries, more interactivity in user interface, etc.

Various Links

- Read about algorithms used for this in industry, academia, in the following links:
  http://norvig.com/spell-correct.html
  http://www.lleess.com/2013/03/auto-correct-algorithm-in-python.html
  http://blog.notdot.net/2007/4/Damn-Cool-Algorithms-Part-1-BK-Trees
  https://www.topcoder.com/community/data-science/data-science-tutorials/using-tries/
  Scientific articles:
  http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/36180.pdf
  http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=52A3B869596656C9DA285DCE83A0339F?doi=10.1.1.146.4390&rep=rep1&type=pdf

Concluding Words #2