CS205 Project Proposal

Implementing Context-Based Optical Character Recognition (OCR) Error Correction in Apache SPARK

Team Members

Kendrick Lo: klo@g.harvard.edu
Gioia Dominedò: dominedo@g.harvard.edu

Overview

The paperless office is for many businesses a lofty goal, representing the achievement of ultimate efficiency and a dedication to environmental sustainability. Despite good intentions, however, certain industries continue to lag behind, including both the legal and finance industries in which we worked.

The naïve approach of *going electronic* typically involves scanning documents as images, for storage. In this regard, Optical Character Recognition (OCR) technology is becoming increasingly important as it allows the content of such scanned documents to be searched. However, automatically detecting errors – both quickly and accurately – remains a challenge.

We canvassed research in the field (see selected references on the right), and discovered that such challenges continue to exist despite the fact that error correction algorithms have existed for some time. We are very interested in exploring these issues further given their practical significance. Furthermore, as proficiency with Apache Spark is a skill that is growing significantly in demand, we are interested in gaining practice and experience with this framework.

Objectives

One article that was of particular interest describes a relatively lightweight Python implementation of a spellchecker, published online by Peter Norvig. Subsequently, Wolf Garbe published an algorithm in C# ("Faroo") that is purported to run orders of magnitude faster than Norvig's approach while maintaining simplicity. Both implementations have been reproduced in several (computer) languages, but none in Spark, to our knowledge. We also think there is further room for improvement, not only by introducing parallelism but also by considering the ***context*** of the individual words being checked.

Our program will take as input a text file with errors, and output words flagged for correction. We will assess how the speed of completion and the error rate (e.g. how many correctly spelled words were flagged for correction, how many incorrectly spelled words were not). Processing will involve two phases:

- **Word-level correction** – implementing the Faroo algorithm, which looks at each word in isolation to determine misspellings.
- **Context-based correction** – looking forward and backwards 1 to 2 words, potentially for multiple passes, to recommend appropriate substitutions given a word's relative position to surrounding words.

References

Wolf Garbe, *1000x Faster Spelling Correction Algorithm* (link1, link2, link3)

Peter Norvig, *How to Write a Spelling Corrector* (link); *Natural Language Corpus Data: Beautiful Data* (link)

Tong et al., *A Statistical Approach to Automatic OCR Error Correction in Context* (link)

Jurafsky et al., *Spelling Correction and the Noisy Channel* (link)

Roger Mitton, *Spellchecking by Computer* (link)

Whitelaw et al., *Using the Web for Language Independent Spellchecking and Autocorrection* (link)

Schedule & Milestones

November 16, 2015
Background reading. Replicate Faroo algorithm in Python, to use as baseline. Compile corpora.

November 23, 2015
Map out specs for program components (e.g. RDD structures). Implement Faroo algorithm (word-level correction) in Spark.

November 30, 2015
Extend algorithm with context-based correction (potentially including build of tri-gram database).

December 4, 2015
Implementing optional features (time permitting). Performance analysis. Generating report, website and video.

More specifically, our algorithm should take into account the frequency of *2-grams* and *3-grams* (*n-gram* refers to a group of *n* consecutive words) that can be generated from the word being checked, and its corrections, within a reference corpus. A proposed outline of our algorithm is provided at the end of this document.

Optional enhancements (time-permitting and pending further consideration of feasibility) might include:

- Pre-processing – looking at the *character-level* to determine whether groups of letters (e.g. 3) are uncommon, suggesting a mutation of small groups of characters, apparently common in OCR-read documents.
- Other adjustments (e.g. considering capitalization or punctuation, the re-occurrence of the (same word) / (groups of characters) / (groups of words) within a portion of the same document that might suggest the word is not misspelled).

Design Overview – Technologies and Use of Parallelism

As previously noted, our error correction program will be written in Spark. We will leverage concepts we learned in the homework problems, including how to optimize performance of the code by reducing unnecessary shuffles.

With respect to word-level correction, the *edit distance* between two words -- the number of edits it takes to turn one word into another -- is important to consider. Significant benefit can be achieved by considering an edit distance of greater than 1, but the computations required increase very quickly. For example, Norvig observes that processing the word "something" generates nearly 500 potential candidates for corrected words with an edit distance of 1, **but this jumps to over 114,000(!) potential candidates for an edit distance of 2**. The Faroo algorithm attempts to decrease the number of potential candidates to be generated during the word-checking phase by performing pre-processing of words in the reference corpus. In any event, we think that parallelizing generation of these words and comparison to the reference corpus of words should enhance performance.

Similarly, with respect to context-based correction, we are considering computing, for each word checked, not only bi-grams that contain the given word or its corrections and comparing them to a database of bi-grams, but also the tri-grams that contain the given word. The number of n-grams to be checked increases as n increases, and **the potential data structures used to search for matches also grows very quickly** (e.g. if we were to use Google's n-gram database, there would be nearly *1 billion* tri-grams to consider). Applying parallelism may afford huge advantages.

Verification

We will compare performance (**in terms of both speed and accuracy**) of our Spark implementation for word-level correction with the serial implementation of the Faroo algorithm (the "baseline") that we will implement in Python. Subsequently, we can compare performance of our program after implementing context-based correction to the same baseline.

Division of Work and Outline of Algorithm

We intend for each member to have input on all program components, but we propose primary responsibility for the word-level correction and the context-based correction components to be assigned to Kendrick and Gioia, respectively.

word-level correction	context-based correction
Input: reference corpus, source text (with errors) 1. process reference corpus to determine counts of all words to obtain our "dictionary" (training) 2. for each word in source text (in parallel): a. enumerate all words of 1-distance in the dictionary b. enumerate all words of 2-distance in the dictionary c. find set of potential corrections sorted by frequency d. suggest word of highest frequency as corrected word	Input: list of top N suggested word-level corrections from 2(c), lists of top 2-grams or 3-grams (or train using reference corpus) 3. for each word in source text (in parallel): a. generate 2 sets of bi-grams (prior word, w), (w, next word) where w is each potential correction from 2(c) b. generate a set of tri-grams of the form (__, __, w), where w is each potential correction from 2(c) c. choose best guess for correct word (e.g. majority vote for most frequent word in each, weighted counts or ??)