DESIGN AND IMPLEMENTATION OF A CONTEXT SENSITIVE SPELLCHECKER USING BAYESIAN AND UNSUPERVISED METHODS

pallavi gudipati & gangal varun prashant

March 20, 2015

Abstract

The aim of the project was to design, build and validate a spell-checking system which would correct out-of-dictionary words. Three different systems were to be built which would perform spell-checking at three levels of granularity - word(in isolation), phrase and sentence. At the word level, we largely relied on Bayesian methods based on counts from the Norvig corpora for ranking, and experimented with two different methods of candidate retrieval. At the phrase and sentence level, we experimented with different methods of capturing the context- such as word ngrams, tagged ngrams, part of speech tag ngrams as well as context words. For performing context-sensitive spelling correction, we also experiment with an unsupervised approach presented in [1], based on ngram counts. The corpora used for learning the context features are the COCA and Brown corpora. All of these methods suffer from varying degrees of sparseness, and we come to the conclusion that a combination of these methods would be adequate.

contents

1 CHOICE OF DISTANCE MEASURE

In order to perform spell correction, we need to have some notion of similarity between the incorrect word, and other correct words, from which we could suggest some as candidate correction. Since similarity requires us to define a distance measure, we need to arrive at a choice of distance measure.

For measuring distance between non numerical objects, we have distance measures based on the idea of edit distance. The edit distance between two objects O_1 and O_2 is defined as the minimum number of edits required to transform the object O_1 into object O_2. The specific definition of what constitutes an edit depends on the domain of objects under consideration. Here, the objects considered are strings.

The traditional version of edit distance between strings is also known as the Levenshtein distance. The three basic edits allowed in Levenshtein distance are insertion, deletion and substitution of a letter. However, for the domain of spell-checking, previous work[4] has noted that the Damerau-Levenshtein(DL) distance, which includes the additional edit operation of transposition(replacing the sequence xy with the sequence yx), is a more efficient distance measure.

2 RESTRICTED EDIT DISTANCE

The restricted edit distance(optimal string alignment algorithm) between two strings is the number of edits using insertion, deletion, substitution and transposition to convert the first into the second, provided no substring is edited more than once. We can clearly see that this is a restricted version of the full DL distance. The full DL distance also allows us to edit a substring more than once.

A key advantage of the restricted edit distance is that it can be computed using a straightforward extension to the matrix-based DP formulation traditionally used for computing the simple Levenshtein distance, while maintaining its $O(n^2)$ complexity. The DP formulation is also of convenience, when we need to compute all shortest paths which result in the edit distance. However, the full DL distance cannot be computed using an extension of the same formulation. Moreover the optimal algorithm to find the DL distance is based on an existential argument, making it difficult to extend it to find all the optimal paths. Also, this algorithm has $O(n^3)$ complexity.

Since we need to compute all the optimal paths during the likelihood computation stage later, we decided to use the restricted edit distance for this step. However, restricted edit distance suffers from the disadvantage of not being a metric. As a consequence, one cannot use index structures designed for metric distances, such as metric trees, to index objects based on this distance. This can affect our options during the candidate retrieval stage. Also, since restricted edit distance is only a loose upper bound on the full DL distance, we may miss out on candidates which are actually at a distance below the desired threshold, if we impose the threshold based on the restricted edit distance.

Hence, we decided to use the full DL distance for the candidate retrieval stage(where we do not need to explicitly compute the edit paths) and the restricted edit distance for the likelihood computation stage.

3 CANDIDATE RETRIEVAL

We considered two candidate retrieval techniques- namely an Inverted Trigram Index and Buckhard-Keller(BK) trees.

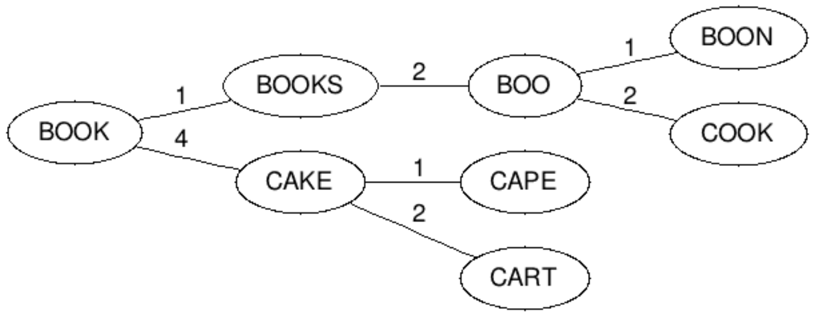

Figure 1: A BK Tree

3.1 *BK-Trees*

We can formulate the task of retrieving candidates as that of retrieving all dictionary words which are within a certain range of DL edit distance from the typo, t. BK-Trees are one of the class of tree index structures known as metric trees, which can index points based on any metric distance measure, and support range queries. Empirically, BK-Trees have been observed to be the best amongst the metric trees for the spellchecking problem. Given a typo t, we can perform a range query on the BK Tree to return all words which are within an edit distance N, which is the tolerance of the range query. Usually, the tolerance is set to a small value such as 2 or 3.

The key idea behind the BK-Tree is to use the triangle inequality, which holds true as a consequence of the distance being metric, to prune candidates during a range query. We then recursively inspect every node. If the node is within the range, we include it in the answer set. We then recursively visit all children of that node, which cannot be pruned away using triangle inequality.

For example, if we searched for the closest match to the misspelling *caqe* within the tolerance of 1 in the above structure we would traverse the tree in this fashion

- *caqe* and *book* are at a distance of 4. Visit all children along edges with weight between 3 and 5.
- Visit *cake*, which is at a distance of 4 from *book*. Since *cake* is at a distance of 1 from *caqe*, include it in the candidate set. Visit all children between 0 and 2.
- Visit *cape*, which is at a distance of 1 from *cake*. Since *cape* is at a distance of 1 from *cape*, include it in the candidate set. *cape* has no children
- Visit *cart*, which is at a distance of 2 from *cake*. Since *cart* is at a distance of 2 from *caqe*, we do not include it in the answer set. *cart* has no children

Empirically, we observed that an edit distance threshold of 2 ends up visiting around 25-30% of the nodes in the tree(which is the size of the dictionary). With a threshold of 3, the number of nodes visited was around 40% to 50%.

3.2 *Inverted Trigram Index*

The trigram-based inverted index essentially queries a table(or an inverted index), which maps every letter trigram to the set of all dictionary words which contain the trigram. For a given typo t, we generate the candidate set by taking the union of all the sets of words returned on querying the inverted index for each of its constituent trigrams.

3.3 *Comparison*

The trigram-based inverted index method suffers from the following disadvantages in comparison to the BK-Tree based method.

- **Low Recall**
  Consider the dictionary word *abcde*, which is misspelt as the typo $abfde$. When we retrieve the candidates for the typo $abfde$, a word w in the dictionary can be a candidate iff it contains atleast one trigram from the set $\{abf, bfd, fde\}$.

  However, *abcde* does not satisfy the above condition. Since it does not appear in the candidate set, we risk missing out on suggesting it amongst the corrections, since it is not even considered for the ranking phase. Thus, we see that a candidate which is just one edit(a substitution of c by f) away from the typo is not a part of the answer set. Hence, this method suffers from low recall.

  In the above example, we demonstrated the low recall of this method when there is just one edit, for a word of length 5. In the setting where there are multiple edits allowed, the degradation in recall is likely to be even worse. This disadvantage is alone enough to render this method unsuitable for this problem.

  Consider the typo *mapon*. On using the BK-Tree based method, the top ranked candidate is *mason*, which seems to be the valid correction in this case. However, on using the trigram-based method, *mason* does not figure in the list of candidates. As a result, the top ranked candidate returned by the word spellchecker is **capon**. Here we see that the low recall of this method can have a direct impact on the correctness of the spellchecker.

- **Spurious Candidates**
  Consider the typo *asociation*, which is generated as a result of misspelling the word *association*. On generating candidates by the trigram based method, the word *nation* will be a member of the candidate set. However, one can intuitively see that *nation* is not a candidate(though there is no ground truth here, the intuition can serve as our ground truth). Hence we are generating a spurious candidate, which will unnecessarily increase the time of the Bayesian ranking step, hence increasing our query time.

  We must, however, note that the latter does not affect the final precision of our results, since a candidate which is actually far away will get ranked down in the Bayesian ranking phase anyway.

The trigram-based method may be advantageous to adopt in comparison to the BK-tree based method in certain cases, however, since it is free from the assumption that all candidates have to lie within a certain range of edit distance from the typo. We illustrate this with an example.

Consider the typo $fotograf$, which is a misspelling of the word *photograph*. The word *photograph* is at an edit distance of 4 from $fotograf$. Hence, if we have a range threshold of 3 on the edit distance(as is commonly used), we only get the candidates *monogram* and *hologram*.

On using the trigram based method, *photograph* is amongst the list of candidates considered, and is ranked as the top candidate in the Bayesian ranking phase.

4 BAYESIAN RANKING

4.1 *Formulation*

Consider a source S, who generates words according to its model of the language. However, the word generated passes through a noisy channel(which in this case models primarily the typographic error), resulting in the typo t. Like in the diagnosis problem, where we attempt to rank the diseases given a symptom, here we will invert the process of generation and attempt to rank the possible correct words which could have led to the typo. Using Bayes Rule, we get

$$P(c|t) \propto P(t|c)P(c) \tag{1}$$

The probability distribution over the correct word, P(c), is also known as the prior or the language model, while $P(t|c)$ is also known as the likelihood or the channel model.

4.2 *Dataset*

We use the dataset by Peter Norvig, which has counts for all the four types of edits, along with frequency counts of words(for the prior model).

4.3 *Dictionary*

In order to decide whether a given word is out of the dictionary, we first need to choose a suitable dictionary. Although Norvig's list of words with their frequency counts is quite vast and comprehensive, it contains words from other languages, slang, names and other words used in colloquial usage (such as Bhattacharya and Nawaz). These words would have introduced a lot of noise into our prior model, while also making it difficult to detect typos which match with non-English words which occur in that corpus. Hence, we decided to use only the words present in the Unix dictionary as the set of correct words, although we took the counts for the same from the Norvig corpus. We filtered out names and abbrievations(which are listed in camelcase) from the Unix dictionary, to prevent problems similar to the ones specified earlier

4.4 *Prior*

The prior probability of a correct word is simply the number of times it occurs in the corpus as a fraction of the total number of words in the corpus.

4.5 *Smoothing*

For smoothing the prior counts, we use the add-one smoothing approach.

4.6 *Likelihood*

[4] defines the Bayesian formulation only in the scenario of a single edit by substituting the likelihood $P(t|c)$ by $P(e)$, where e is the single edit which transforms c into t. However, since we are now working in a setting where multiple edits are allowed, we need to modify this formulation.

Consider a typo t, attained as a result of multiple edits from the correct word c. Let these edits be e_1, e_2, e_3 and so on. Then the likelihood can be written as

$$P(t|c) = \prod_{i=1}^{i=L} P(e_i) \tag{2}$$

where L is the length of the edit sequence, which we also refer to as an edit path. However, we must note that there can be multiple edit paths, all having the same length equal to the edit distance between t and c, which can lead to the edit. We then need to sum up the probabilities of all these edit paths. In other words,

$$P(t|c) = \sum_{i=1}^{i=P} P(p_i) \tag{3}$$

$$= \sum_{i=1}^{i=P} \prod_{j=1}^{j=L_i} P(e_{ij}) \tag{4}$$

4.7 *Evaluation*

We evaluated our word spellchecker on the Wikipedia corpus, consisting of 4261 commonly misspelt words. We experimented with both methods of candidate retrieval. The performance was as follows

Method	Incorrect Answers	Accuracy	Time Taken
BK Tree	245	0.9425	8-10 minutes
Inverted Trigram Index	426	0.9000	2.5 hours

We see that the inverted trigram index compromises significantly on accuracy as well as time.

5 PHRASE SPELL-CHECKER

We build upon our word spellchecker to get our phrase spellchecker. We add two language models to our word spellchecker, Context Words model and NGram model. We haven't used part-of-speech(POS) tags in our phrase checker as phrases don't have a concrete structure. However, we will include them in our sentence spellchecker.

5.1 *Candidates*

To generate candidates for phrase spellchecker:

- Traverse through the phrase and get all the word errors.
- Generate ranked candidates for each misspelled word using word spellchecker.
- Depending on the number of misspelled words in the phrase, take top-k candidate corrections for each of them.
- Exhaustively enumerate through the selected corrections and generate phrase candidates.

After generating candidates, we pass the candidates through both Context Words model as well as NGram model. We combine the scores obtained through both the language models with the edit distance score that was computed in the candidate generation phase.

5.2 *Datasets*

5.2.1 *NGram Corpus*

We use take our ngram counts from the Corpus of Contemporary American English(COCA). This corpus has ngram counts for n upto 5. We chose this corpus since it fits into main memory, unlike other larger corpora such as the Google NGram Corpus, which cannot be handled except on a large scale system. The COCA corpus has the following number of distinct ngrams

- 1020385 bigrams
- 1020009 trigrams

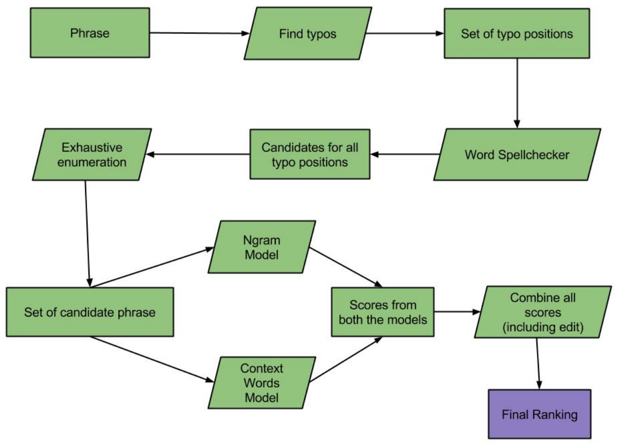

Figure 2: Phrase Spell-checker Approach

- 1034307 quadrigrams
- 1044268 pentagrams

5.2.2 *Brown Corpus*

The Brown Corpus consists of 500 samples, distributed across 15 genres in rough proportion to the amount published in 1961 in each of those genres. All works sampled were published in 1961; as far as could be determined they were first published then, and were written by native speakers of American English.

Each sample began at a random sentence-boundary in the article or other unit chosen, and continued up to the first sentence boundary after 2,000 words. In a very few cases miscounts led to samples being just under 2,000 words.

We parsed the Brown corpus to generate context words as well as POS tags.

5.3 *NGram Model*

For learning from the ngram model, we adopt $SUMLM$, an unsupervised approach first proposed in [1]. This method produces a score for every candidate-substituted phrase, by adding up the log-counts of all ngrams occuring in the sentence. Intuitively, we can see how this approach makes sense. If a substituted phrase A has ngrams which occur more frequently than another substituted phrase B, then the former is a better correction than the latter.

$$W(S) = \sum_{i=1}^{i=n} log(c(w_i)) \tag{5}$$

Here, $W(S)$ is the score of the substituted phrase, and each w_i is an ngram occurring in the phrase, with $c(w_i)$ being its count. Although it may seem incorrect to include all the ngrams, since some of them may not span any position at which we perform a correction, these ngrams only add a constant offset to every phrase in the set of candidate phrases, and hence do not affect the final ranking. Moreover, the number of ngrams in a sentence is linear in the size of a sentence, hence the complexity of finding the score for a substituted phrase of length L is $O(L)$.

[1] claims that $SUMLM$ is similar to a Naive Bayes classifier, but without counts for the class prior. Moreover, the results in [1] demonstrate that $SUMLM$ performs almost as well as its unsupervised variants.[1] considers the application of $SUMLM$ to the context sensitive spelling correction problem with confusion sets. However, here we apply it for the non-word to word

correction problem, generating all substituted phrases by considering the set of top 20 ranked candidates(as per our word spellchecker) as the confusion set at each position.

5.4 *Context Words Model*

Another way of modelling the context of a word c is the bag of words approach. We can represent the context C as the set of words occurring in the locality of c. These words are also known as context words.

$$P(c|C) = P(c|c_1, c_2 ... c_K) \tag{6}$$

To overcome sparseness in the data, we decide to make the Naive Bayes assumption of conditional independence between the context words, as outlined in [3].

$$P(c|C) = \prod_{i=1}^{i=K} P(c|c_i) \tag{7}$$

We set the size of our context window to 6(3 words before and 3 words after the occurrence). This was empirically found to be the best size, as specified in [3].

6 SENTENCE SPELL-CHECKER

In addition to two language models used in phrase spellchecker, we also add another language model(”TagGram” model) that is based on POS tags. The POS tag corpus is generated using the Brown corpus. We have also used Stanford’s Log-linear Part-Of-Speech Tagger to tag the candidates.

6.1 *Stanford Log-linear Part-Of-Speech Tagger*

This POS tagger was developed by The Stanford Natural Language Processing Group. We use the existing English model $english - bidirectional - distsim.tagger$ and train it with the data provided as a part of the package, $english - bidirectional - distsim.tagger.props$. We then query the tagger for the POS tags of our candidates.

6.2 *Different POS tagsets*

We have used Brown corpus for POS tags data and Stanford’s POS tagger to tag the candidates. However, both of them use different tagsets. Thus, we have reduced both of these tagsets to a new and a much simpler tagset. This tagset contains only 14 different tags. This groups lots of categories under one tag. But this also reduces our accuracy as it makes the tagger more coarse-grained. The whole mapping can be found at `http://goo.gl/AgtlG8`.

6.3 *The TagGram Model*

We refer to a n-gram of only POS Tags(without the word) as a TagGram. A TagGram is essentially a sequence of POS Tags. For instance, the sequence PREP DET NN is a taggram of length 3.

In [2], the authors had demonstrated that part-of-speech trigrams could be effectively used when combined with the context words approach for the context sensitive spelling correction problem. This serves as our motivation to use TagGrams to enhance our language model.

However, the paper only uses trigrams, whereas we decided to use all ngrams between the range of 2 to 5 which occur in the context of the occurrence. We again use the $SUMLM$ method to weigh the candidates according to their TagGram counts, as we had done earlier in our NGram model.

6.4 *Weighing the Models*

In the final score for every candidate phrase/sentence, we sum up the log weights for the edit component, and the various language models, giving equal weightage to each. This is equivalent to taking the simple product in the original probability product. In the absence of a considerably large validation set to learn weights on each of these components, we decide to assign equal weights (in essence taking the unweighted sum of log probabilities).

REFERENCES

[1] S. Bergsma. Web-scale n-gram models for lexical disambiguation. *IJCA*, 2009.

[2] A. R. Golding. Combining trigram-based and feature-based methods for context-sensitive spelling correction.

[3] A. R. Golding. A bayesian hybrid method for context-sensitive spelling correction. *arXiv preprint cmp-lg/9606001*, 1996.

[4] M. Kernighan. A spelling correction program based on a noisy channel model.

[5] Nlp.stanford.edu. The stanford nlp (natural language processing) group, 2014.