Natural Language Processing
Spell Check Assignment

Abhishek Narwekar (EE11B129)
Deepak Mittal (CS15S014)
Mahesh Balan (MS14S012)
Team Number : 21

4th October 2015

1 Word Spell Check

1.1 N-gram based Similarity (N-SIM)

1.1.1 Definition

N-gram based measures, albeit in the word n-gram sense, have been used for tasks such as text classification [1], and has been shown to work well independent of language [2]. We define an N-SIM basic similarity measure between two words which is mathematically easy to compute and intuitive. Treating each letter in a word as a unigram, we first enumerate all the consecutive subsequences of size N that lie in the word. Let us denote the set of N-grams as S_1 and S_2 respectively for words w_1 and w_2. It is easy to see that the cardinalities of these two sets are $(|w_1| - N + 1)$ and $(|w_2| - N + 1)$ respectively. There exist multiple similarity measures between two sets. One of the most common ones is the Jaccard Similarity, given by:

$$Sim(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cap S_2|} \tag{1}$$

1.1.2 Details about Computation

In order to reduce runtime, we perform an offline computation and storage of an inverted index of N-grams. Essentially, we break each word in the dictionary into N-grams, and also keep a track of the words in which each N-gram occurs. During execution, we first load these forward and inverted indices. Given a test word w_{test}, we divide it into N-grams for values of N we are interested in. For each N-gram belonging to the word, we use the inverted index to obtain the words in the dictionary containing that N-gram. Repeating over all N-grams, we get a *set of candidate words* S_C which are similar to w_{test}. For each candidate

$w \in S_C$, we compute the Jaccard similarity $Sim(w_c, w_{test}9)$. Finally, we sort the candidates in decreasing order of similarity and choose the top-K as required.

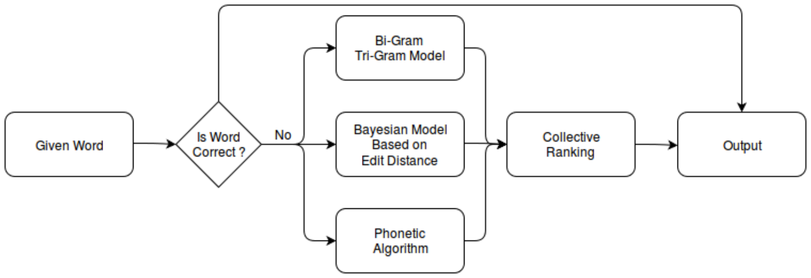

Figure 1: Word Spell Checker: A Block Diagram

1.1.3 Advantages and Drawbacks

The edit distance between two words is defined as the smallest number of operations from a set of operations that must be performed to transform one word into the other. Most similarity measures depend strongly on the edit distance, and tend to stop at edit distance 2 or 3, owing to a nearly exponential growth of the search space. One major advantage of N-SIM is that it doesn't *explicitly* depend on the edit distance between w_{test} and words in the dictionary. Of course, the larger the edit distance to a word w_0, the less likely it is to be similar to w_{test}, but at least we recover all words having at least one N-gram in common with w_{test}. For small values of N such as $N = 2$, the recall of this measure is good. For instance, consider the misspelt word *thruout*. Although the intended word, *throughout*, is at an edit distance of edit 3 from it, we can recover it as a candidate word since it has a good Jaccard similarity with the misspelt word:

$$Sim_2(thruout, throughout) = \frac{|\{th, hr, ou, ut\}|}{|\{th, hr, ro, ou, ug, gh, ho, ut, ru, uo\}|} = 0.4 \quad (2)$$

In the above equation, the subscript 2 in $Sim_2()$ indicates that it is a bigram similarity measure. However, a drawback of n-gram based similarity is its low precision. For low values of N in particular, the process of candidate retrieval may bring in a lot of spurious candidates, such as the case of *outthrust* and *thrutch* being ranked as the two most similar words to *thruout*. As a result, we need to increase the order, N, of the N-gram. This empirically improves the precision of the spell-checker but lowers the recall. For instance, if the word *spark* is misspelt as *spork*, N-SIM with $N = 3$ will not consider these two words similar, since they do not have a single trigram in common. Thus, there is a tradeoff betweem the precision and the recall for different N. For smaller words, a lower order N-SIM tends to work better.

1.2 Bayesian Estimation

1.2.1 About the Model

We implemented the noisy channel model based Bayesian spelling correction as proposed by Kernighan et al [4]. In this model, we try to maximize the conditional probability of a correct word given the incorrect word. Essentially, we try to answer the question: what is the likelihood that the writer had a (correct) word X in mind when he wrote X'? Mathematically, we can express this problem as:

$$c_0 = \underset{c \in D}{\operatorname{argmax}}\{P(c|w)\} = \underset{c \in D}{\operatorname{argmax}}\{P(w|c)P(c)\} \tag{3}$$

Here, D is the dictionary and w is the incorrect word. By virtue of the noisy channel model, we can model the process of going from w to any $c \in D$ as an effect of noise. This noise manifests in four possible ways: insertion, deletion, substitution and transposition. Using them, we can compute what is known as the Damerau–Levenshtein distance between two words. The probability of each form of noise (hereby referred to as an "edit") is dependent on source and target alphabets, resulting in *confusion matrix*. Thus, the probability of each possible edit can be obtained from the confusion matrix corresponding to the edit. So now, we can model each change as a sequence of edits $\mathbf{E} = [E_1, E_2 ... E_n]$. Thus, from equation [3], we obtain:

$$P(c|w) = P(\mathbf{E}|c)P(c) \tag{4}$$

This probability is tough to find, since each edit can take a huge number of values. Thus, we need a large enough dataset to avoid running into sparsity problems. Therefore, we make the Naïve Bayes assumption here, rendering all the edits independent of each other. Thus, the initially difficult-to-compute probability of edits breaks down into a product of terms. We must also consider the case when a word c may transform into w through multiple *sets* of edit sequences, $\mathbf{E^{set}} = \{\mathbf{E_1}, \mathbf{E_2} ... \mathbf{E_k}\}$.

$$P(c|w) = \sum_{E \in \mathbf{E^{set}}} \prod_{e \in E} \{P(e)\} P(c) \tag{5}$$

1.2.2 Computational Details

Since evaluating the probabilities of edits from all possible words in the dictionary to the incorrect word is a very expensive process, we limit ourselves to those words in the dictionary which are at an edit distance of 2 from the incorrect word. To do so, we first compute the set of all the words $\mathbf{S^1_w}$ that can be derived from the incorrect word using just 1 edit. We also keep track of the changes that we make to arrive at words in $\mathbf{S^1_w}$. To get the set of words $\mathbf{S^2_w}$ at an edit distance of 2, we compute all words at an edit distance of 1 from every word in $\mathbf{S^1_w}$. Here too, we keep track of the edits that we make. We stop here, and hypothesize that sequences of larger edits have a negligible probability of occurrence as compared to the sequences in $\mathbf{S_w^1}$ and $\mathbf{S_w^2}$. To generate a set of candidate words C_w, we simply take the set of words in $\mathbf{S_w^1} \cup \mathbf{S_w^2}$ that lie in the dictionary. Let us denote this set of candidates as $\mathbf{S_c}$. For each candidate word in $c \in \mathbf{S_c}$, we know the sequence of edits that a correct word undergoes to arrive at the wrong word w. Thus, we can, using the confusion matrices,and the expressions proposed by Kernighan et al [4], compute the a posteriori probability of the correct word given the wrong word.

1.2.3 Smoothing

Given that each probability term is small (of the order of 10^{-5}), a product of many such terms can result in an underflow. Thus, we compute and maximize the log likelihood:

$$c_0 = \operatorname*{argmax}_{c \in D} \sum_{i=1}^{n} logP(E_i) + logP(c) \tag{6}$$

To account for those edits which don't occur in the training set, we perform a add-one Laplacian smoothing on the entire set of training data. Essentially, we add one to each element of every confusion matrix. This prevents the problem of underflow and penalizes those terms in the test data which have not been seen before.

1.3 Phonetic Similarity

There are cases of words like *thruout*, which are phonetically similar to the word *throughout*, but are far apart in terms of edit distance. The method of N-gram based may not perform reliably in such cases, as seen in section 1.1.3. If we can obtain the words phonetically similar to the incorrect word, we can propose newer and better candidates. The most widely known algorithm for a phonetic mapping, Soundex, [7], is a rule based algorithm. A more recent improved algorithm, Double Metaphone [8], incorporates more sophistication and is known to give better accuracy. It gives us two representations - a primary and a secondary - of the input word. However, in our case, the algorithm exhibited a rather low recall. There was a large many-to-one mapping from the words in the dictionary to their phonetic representations. Thus, although *thruout* and *throughout* were both mapped to *ORT* using Metaphone, so were *dirt*, *toured*, *trat* and *tardier* among others. Thus, this measure didn't perform so well for the task.

2 Phrase Spell Check

In the Phrase Spell Check problem, we have to additionally use the context around the incorrect word to infer the correct word. To perform this task, we implemented a hybrid of two models: Context Words Based Estimation and Word Splitting.

2.1 Context Words Based Estimation

In look around the incorrect word a certain predefined window to obtain the context. For example, with a window size of 2, consider the following sentence:

Chocolates are ***tasteir*** *than candies.*

In the above sentence, the word ***tasteir*** is incorrect. So the context words around it will be {*Chocolates, are , than, candies*}. Using this context, we perform the spell check, as described in the following section. We used the Brown Corpus for training our Spell Check Model. We follow the context based model proposed by Golding [9] here.

2.1.1 Procedure for Training

Firstly, we store the distribution of words around each word using the Brown Corpus. This includes the distribution of counts of words around each word in the dictionary. Offline computation of this distribution saves us runtime. We also store the likelihoods of each context word in the data upon add-one smoothing. The likelihood term thus looks as follows after smoothing:

$$P(w|c) = \frac{1 + freq(w|c)}{\sum\limits_{x \in V} freq(x|c) + |V|} \tag{7}$$

Here, $freq(w|c)$ is the number of times a word w occurs in the context of a word c in the training data and V is the vocabulary. We observed that setting the smoothing factor to lower values gave better results, since they penalized non-occurrences.

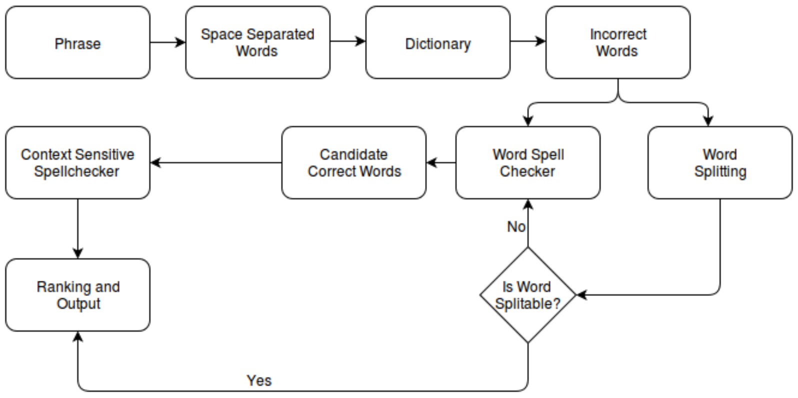

Figure 2: Phrase Spell Checker: A Block Diagram

2.1.2 Procedure assuming at least one error

We first detect words in the test phrase which are not in the dictionary. Such words form our candidates for spell check. For each candidate thus generated, we use the methods described in section 1 to obtain a suggestion set $\mathbf{T}$ for the incorrect words thus detected. Now, our task is to evaluate the likelihood of each suggested word given the incorrect word. Given the context words $\mathbf{C} = \{c_{-k}....c_{-1}, c_1...c_k\}$ in a window of size k, the likelihood of the correct word given the context is given by:

$$P(c|\{c_{-k}....c_{-1}, c_1...c_k\}) \propto P(\{c_{-k}....c_{-1}, c_1...c_k\}|c)P(c) \tag{8}$$

Here, c is the correct word. Using the Bayesian approximation in section 1.2, we say therefore that:

$$P(c|\{c_{-k}....c_{-1}, c_1...c_k\}) \propto \prod_{w \in C} P(w|c)P(c) \tag{9}$$

Following this, we follow equation [6] to convert the product to a sum through a $log()$ operation. We choose that candidate term c_0 which maximizes the likelihood.

$$c_0 = \underset{c \in \mathbf{T}}{\operatorname{argmax}} \sum_{w \in \mathbf{C}} logP(w|c) + logP(c) \tag{10}$$

2.1.3 Procedure Assuming No Errors

Assuming no spelling, errors are detected, one possible way to perform disambiguation is too look at some commonly misspelt words and see if any of them match, as done in [9]. However, that may not cover a lot of possible errors. Thus, the method we adopt is as follows: assume that each word can be at fault, and get a set of candidates for all words. For each word, we choose the likeliest word and form the sentence.

2.1.4 The Prior Term

We observed that there exists a sparsity in the Brown Corpus with respect to the context words. Consider for example the classic “desert” vs “dessert” confusion. After including obvious context words such as “sand”, “camels”, etc. around “dessert”, the difference in the likelihoods is large. This is expected, since the context is doing its job of giving us a better likelihood. This is illustrated in table [1].However, if a word like “in” is suggested to be “is”, there is very little difference between their likelihoods, and in many cases, “in” may get confused with “is”, especially if there is no spelling error in the sentence. Thus, the prior term gains huge importance in such cases. We observed that if a word is not incorrect, assuming a prior of 1 from the word spell check code helps in boosting

	“camels in the dessert sand”	“camels in the desert sand”
$logL(desert)$	-46.60	-28.27
$logL(dessert)$	-50.55	-70.84

Table 1: Likelihood values for two confusion set words

2.2 Word Splitting

To handle cases such as two words merging together, we implemented a splitting algorithm to try and split the given incorrect word into as many legitimate words as possible. For instance, “howareyou” is to be split as “how are you”. This problem involves dividing the input into string into contiguous substrings such that each of them is a word. We implemented a solution for it:

1. Place the divider at the beginning of the string.
2. Move the divider towards the right till the substring to the left of the divider is a legitimate word.
3. If right substring is non-empty: Add this word to the list of words obtained from this string, and with the right substring as the new starting point, repeat from 1
   If right substring is empty and the left substring is also empty: We have a legitimate division If right substring is empty but the left substring is not: We do not have a legitimate division

This procedure is repeated from the other end too, i.e. keeping the pointer at the end of the string and moving in progressively. If we end up with exactly one legitimate division, we return it. If there are two divisions, we add the priors of both the divisions and return the one with the greater sum. The logic is that a division into more common words should be favoured.

2.3 Merging the Bayesian method with Word Splitting Method

We observed that the word splitting method would output false positives at times. For instance, “restaurant” would get split as “re st au rant”, which is a valid phrase. However, this would result in some actual spelling error being overlooked. We worked around this problem by finding an incorrect word using the Bayesian method first, suggesting two recommendations for it, and supplying this word to the splitting method. The output of the splitting method is ranked last.

3 Sentence Spell Check

Since phrases don’t follow a strong grammatical structure, we can not use POS Tags model there. However, in the Sentence Spell Check we implemented an additional model apart from the phrase models: POS Trigram Based Estimation.

3.1 POS Trigram Based Estimation

One drawback of the Bayesian model is that doesn't maintain the ordering of the context words. N-gram based models help us retain the order also while predicting the correct word. We extracted simplified POS tags[10] words in the for Brown Coprpus. For each word in the corpus, we stored the distribution of the N-grams around it. After this, we implemented the maximum likelihood method which we followed for phrase spell check. We experimented with window sizes equal to {1,2,3}. As expected, the best performance was for window size = 1. This can be explained by the observation that the sparsity in our data increases with an increase in window size. For larger N-gram based methods, we would need larger datasets. Further, we also experimented by taking POS Tags which are only left to the word, only right to the word, and on both sides of the word. We observed that taking POS tags on both sides gave the best performance and taking POS tags only on the right side gave the worst performance.

3.2 Merging the POS Trigram method with Phrase Spell Check

We formed a unified list of three suggestions for sentence spell check as follows. We took the first two suggestions from the POS trigram method, and the last one from the phrase spell check method such wasn't recommended already by the POS trigram method.

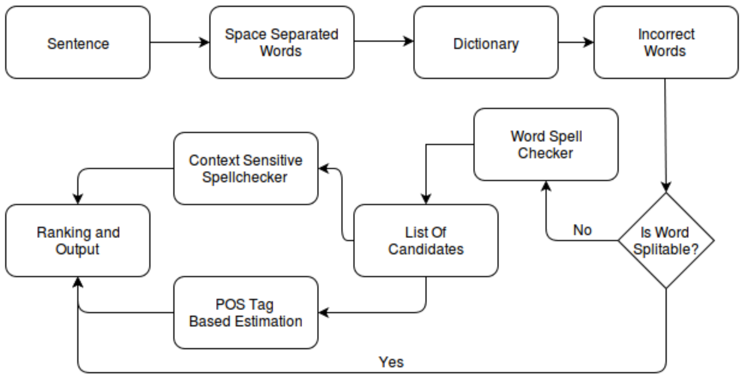

Figure 3: Sentence Spell Checker: A Block Diagram

4 Datasets

For all our experiments we used the Brown Corpus[6] and datasets provided by Peter Norvig[5]. Initially we trained all our models by using ”big.txt” (a corpus provided by Peter Norvig for spelling corrections). But models were not performing well. We observed a noticeable difference upon switching to the Brown Corpus. We believe that the reason for this behaviour could be that although ”big.txt” is larger than Brown Corpus but it contains less than half the number of unique words that are there in Brown Corpus. Moreover, Brown Corpus exhibits more diversity in terms of genre

References

[1] William B. Cavnar and John M. Trenkle, 1994 N-Gram-Based Text Categorization.

[2] Mark Damashek, 1995. Gauging Similarity with N-grams: Language Independent Classification of Text. Science, New Series, Vol. 267, No. 5199 (Feb. 10, 1995), pp. 843-848

[3] Grzegorz Kondrak, 2005. N-gram similarity and distance. String processing and information retrieval

[4] Mark D. Kemighan, Kenneth W. Church, William A. Gale, 1990 A Spelling Correction Program Based on a Noisy Channel Model. COLING ’90 Proceedings of the 13th conference on Computational linguistics - Volume 2

[5] Natural Language Corpus Data: Beautiful Data. Peter Norvig’s webpage.

[6] W. N. Francis and H. Kučera Brown Corpus Manual. A Standard Corpus of Present-Day Edited American English, for use with Digital Computers (Brown). 1964, 1971, 1979. Compiled by . Brown University.

[7] Robert C. Russell and Margaret King Odell, 1918. The Soundex Algorithm,

[8] Lawrence Philips, 2000. The double metaphone search algorithm. A Standard Corpus of Present-Day Edited American English, for use with Digital Computers (Brown). C/C++ Users Journal archive Volume 18 Issue 6, June 2000 Pages 38 - 43

[9] Andrew R. Golding, 1996 A Bayesian hybrid method for context-sensitive spelling correction00.

[10] Simplified POS tags using NLTK.