A major functionality of the integrated text editor is the detection and correction of spelling errors. Given a misspelt word, we would like our editor to detect the spelling error as well as replace it with the most viable candidate word. At the outset, let us first discuss the types of spelling errors that we may come across.

There are two types of spelling errors that we have to deal with:

1. Non word spelling error
2. Real word spelling error.

Non word spelling error:

A spelling error that is not found in a dictionary is termed a non-word spelling error.

Eg: graffe for giraffe

Real word spelling error:

A spelling error that is a valid word found in a dictionary is a real word spelling error. These spelling errors necessitate the inspection of the context in which the misspelling occurs for its detection and subsequent correction.

Eg: desert for dessert and vice versa

Non-word spelling error detection and correction

Obviously, detection of non-word spelling errors isn’t a very difficult task. With a sufficiently large and dependable corpus (dictionary), any word that is not found in the dictionary can be safely identified as a non-word spelling error.

Correction however is slightly more involved. To understand how to implement an algorithm that corrects a non-word spelling error with a viable candidate, it would help to first examine how we humans are able to correct non-word spelling errors. The word graffe is obviously meant to be giraffe, but how did we figure this out? The fundamental reason for such intuition is our ability to identify associations between similarly spelt words. We hence have to quantify this intuition of association between words in order to develop an efficient non-word spelling error correction algorithm. In order to do this, we explore a quantity known as edit distance.

Edit distance

Edit distance is a way of quantifying how dissimilar two words are to one another by counting the minimum number of operations required to transform one string into the other. Hence lesser the edit distance between two words, closer they are to each other in terms of our intuition of association. Common edit operations include insertion, deletion, substitution and transposition. These operations could be assigned different weights while computing the edit distance between two words. We however compute the edit distance after assigning equal weights to all the aforementioned edit operations namely the Levenshtein edit distance, where each operation has an equal weight of 1.

Eg:

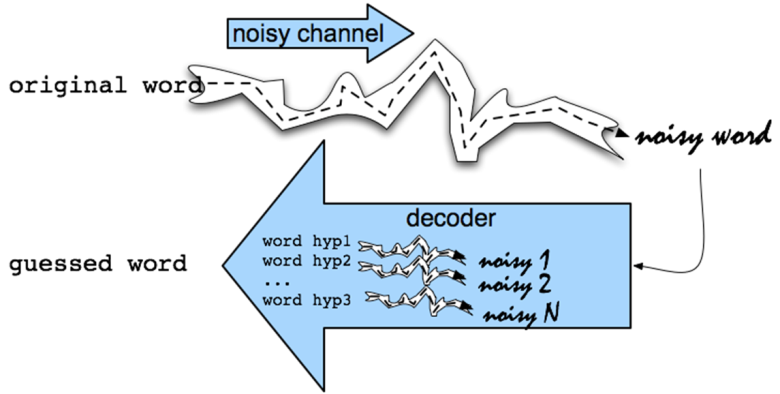
The Levenshtein distance between "kitten" and "sitting" is 3.

1. kitten → sitten (substitution of "s" for "k")
2. sitten → sittin (substitution of "i" for "e")
3. sittin → sitting (insertion of "g" at the end).

We now have to generate all words that are a certain edit- distance away from a given target word (word recognized as a non-word spelling error). The literature on spelling correction claims that 80 to 95% of spelling errors are an edit distance of 1 from the target. We will also take into consideration words that are at an edit distance of 2 from the target word.

Hence upon detection of a non-word spelling error, we generate a set of valid words that are 1 or 2 edit-distances away from the target word. Now we have to pick the most ‘viable’ candidate word that belongs to this set of words. Hence there is a need to enumerate the different candidate words that belong to this set. This is where we introduce the noisy channel model for spelling correction, a model that is largely used for this task.

Noisy channel model for spelling correction



In this model, the goal is to find the intended word given a word where the letters have been scrambled in some manner (scrambled to form a misspelling in this case). Essentially we have a word c that is affected by noise (spelling error) in a noisy channel (the typist) to produce the misspelling w.

Our task is to correctly produce c, given w. This is done by applying conditional probability.

Formally, let C be the set of all candidate words produced given a misspelling w (this is done by producing a set of all words that are at an edit distance of 1 or 2 away from w). Our task is to select c ϵ C such that p(c|w) is maximium.

The probability *p(c|w)* is given by :

Since *p(w)* is the same for every possible c, we ignore it and obtain:

……………………………………………………………..Equation 2

We hence have to find c ϵ C such that p(c|w) given by equation 2 is maximized.

A look into each term in Equation 2 and its intuition will give us an idea of what this implies.

p(c|w) Probability that the typist meant to type c, given he/she has typed w.

p(w|c) Probability that a typist may type w when he/she meant to type c.

p(c) Probability of occurrence of c in a corpus.

p(c|w) hence is directly proportional to p(w|c) and p(c).

In our model we evaluate p(w|c) based on a simple assumption: lesser the edit distance between c and w where c ϵ C, higher is p(w|c). This means that if we find a c ϵ C that is an edit distance of 1 away from w, the corresponding p(w|c) is higher than that for those c ϵ C that are an edit distance of 2 from w. We then resolve conflict between those c ϵ C that are the same edit distance away from w by considering p(c). The c with highest p( c) (having higher frequency in the corpus) is then chosen. The p(c) component makes sure that we don’t suggest obscure words as corrections even if their edit distance from w happens to be the least.

This intuitively means, given a non-word spelling error, we choose the word with highest similarity in terms of spelling to the misspelt word, provided it is fairly common in our corpus.

Now that we have examined the principle behind our spelling error detection and correction mechanism, we can look at the finer implementation details.

General Methodology

1. Once a user types a word w, we search for the same within our corpus. If found, we don’t do anything. If the word isn’t found, we detect w as a non-word spelling error.
2. Once a typed word w is detected as a non-word spelling error, we generate C, the set of all words that are of an edit-distance of 1 or 2 from w.
3. We reduce C to contain only those words that are contained in the corpus.
4. We then choose c ϵ C such that its edit distance from w is least and its corresponding frequency of occurrence in the corpus is highest.

Flowchart