Published in final edited form as:
Inf Retr Boston. 2006 November ; 9(5): 543–564.

SPELLING CORRECTION IN THE PUBMED SEARCH ENGINE

W. John Wilbur, **Won Kim**, and **Natalie Xie**
National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD, U.S.A.

Abstract

It is known that users of internet search engines often enter queries with misspellings in one or more search terms. Several web search engines make suggestions for correcting misspelled words, but the methods used are proprietary and unpublished to our knowledge. Here we describe the methodology we have developed to perform spelling correction for the PubMed search engine. Our approach is based on the noisy channel model for spelling correction and makes use of statistics harvested from user logs to estimate the probabilities of different types of edits that lead to misspellings. The unique problems encountered in correcting search engine queries are discussed and our solutions are outlined.

Keywords

noisy channel model; user query logs; nonword error detection; trie; edit distance

INTRODUCTION

A number of studies of search engine queries have observed a high misspelling rate (Nordlie 1999; Spink, Wolfram et al. 2001; Wang, Berry et al. 2003). Wang, et al. (2003) report a misspelling rate of 26% for words on an academic site. It seems possible that the misspelling rate on a public site could be even higher. Nordlie (1999) observes that two thirds of initial requests are unsuccessful in meeting their objective and an NPD survey (2000) finds that 77% of the time an initially unsuccessful search is modified and tried again on the same site. These findings suggest the potential benefit of performing some kind of query correction for the user. Spelling correction is an obvious candidate for this role. We therefore undertook to study how such a facility could be constructed for the PubMed search engine. PubMed, a service of the National Library of Medicine, provides access to over 16 million MEDLINE citations back to 1950 and additional life science journals (McEntyre and Lipman 2001).

Spelling correction has been a topic for research for many years and the problem has been usefully divided into three subtasks (Kukich 1992; Jurafsky and Martin 2000) in increasing order of difficulty: 1) nonword error detection; 2) isolated-word error correction; and 3) context-dependent error correction. Each of these tasks is relevant to the problem of spelling correction in a search engine and each task is subject to some special considerations in that setting. Nonword error detection has typically been done by comparing a string to the list of accepted words in some dictionary. In the setting of a search engine the vocabulary potentially accessible by a search serves the purpose of a dictionary. For the purposes of this paper let us refer to this vocabulary as the database vocabulary. If a term is not in the database then it can be assumed to be misspelled for the practical purpose of searching the data. If the term is simply of low frequency in the database, it may still have a high probability of being a misspelling

Correspondence to: W. John Wilbur, National Library of Medicine, Bldg. 38A, Rm. 6S606, 8600 Rockville Pike, Bethesda, MD 20894, U.S.A. Phone 301-435-5926. Fax 301-480-2288. Email wilbur@ncbi.nlm.nih.gov.

and we may benefit the user by suggesting a higher frequency term as a correction. If the query is a single word we are dealing with a case of isolated-word error correction. If on the other hand the query consists of two or more words, the possibility exists that we are dealing with a useful context that may aid the correction process. However, queries are usually not more than two or three words (Silverstein and Henzinger 1999) so the context will be at best small and at worst not helpful. In this situation a strategy must be developed to make use of the context where it is helpful and to ignore it otherwise. A typical and practical approach to make use of context in spelling correction is to apply a language model to the genre of text at issue and use it to enhance the prediction of the corrected string (Church and Gale 1991; Kukich 1992; Brill and Moore 2000; Jurafsky and Martin 2000). Our approach is similar to a language model in that when we are presented with a query of more than a single word we attempt to correct to a phrase that is recognized by the query engine and the frequency of that phrase comes into play in the process.

Our basic approach is a form of the noisy channel model for spelling correction that is very similar to the method developed by Church and Gale (1991). The main difference is our inclusion of a letter of context on either side of a putative correction when computing its probability. In this we are moving in the direction taken by Brill and Moore (2000) only we do not allow as much context as their approach. The noisy channel model seeks to evaluate the expression

$$\arg\max_{w} P(s \mid w)P(w) \tag{1}$$

where *s* represents a string to be corrected and *w* a potential correction. In our implementation *w* runs over the database vocabulary of the search engine and *P*(*w*) represents the probability that a user would have intended to search using the word *w*. We follow Church and Gale (1991) in evaluating $P(s|w)$ as the product of the probabilities of the edits required to convert *w* into *s*. One of the difficulties in constructing our correction algorithm has been to obtain useful context dependent estimates of these edit probabilities. Our solution involves harvesting the statistics from the search engine logs.

The paper is organized into the following sections:

- **Gathering the edit statistics** – How we mine edit probabilities from the PubMed user query logs.
- **Basic assumptions of the method** – How we interpret the noisy channel model in the PubMed setting.
- **The algorithm: basic functions** – Four basic editing functions that are applied to strings depending on their characteristics.
- **The algorithm** – How the basic editing functions are combined to process strings of one, two, or more tokens.
- **Cleaning up the PubMed data** – How we down grade incorrect spellings in the PubMed search engine vocabulary using statistical testing.
- **Performance issues** – Figures describing the current implementation of the algorithm and its performance.
- **Discussion** – Successes and failures of the algorithm and how it might be improved.
- Conclusions.

Before we go further just a word about terminology. By the terms “word” or “token” we will mean the same thing, namely, a string of printable ASCII characters not including any white space within the string. The terms “word” or “token” are commonly used interchangeably (Jurafsky and Martin 2000). Thus “house” is a word or a token and so is “xxxxx” though we may not usually think of “xxxxx” as a word. We will also use the words “term” and “phrase” interchangeably to mean a string consisting of one or more words or tokens separated by white space. Again this is common usage.

GATHERING THE EDIT STATISTICS

While spelling correction has not been the focus, a number of investigators have examined methods for mining user query logs for web search engines with the objective of making useful suggestions to improve a user’s query. Beeferman and Berger (2000) cluster queries based on “click through” data that shows which records a user actually selects. When different queries result in the same record being clicked that counts for similarity between the queries. Wen, et al. (2002) use “click through” data as well as a metric on the lexical similarity of two queries for the same purpose. Such methods can be used to suggest terms from one query to supplement a query that has been found close to it in “click space”. Leroy, et al. (2003) use the text “clicked on” rather than the record “clicked to” and mine the words in the text clicked on as a source to supplement user’s queries. Huang, et al. (2003) mine the pairs of terms that co-occur in a single user’s session in web logs to discover relationships that may be used to suggest new terms to add to a user’s query. While none of these studies have our goal of spelling correction there are yet some similarities of note.

We mine a query log to discover single user sessions containing pairs of terms which we identify as a query term and its correction. A single user session is defined by a single IP address and a query term and its correction must occur within 300 seconds of each other. The 300 second threshold has been found useful (Silverstein and Henzinger 1999; Huang, Chien et al. 2003). Data shows that few query pairs separated by more than 300 seconds come from the same session (Huang, Chien et al. 2003). We mine these pairs of query terms to be used not as a direct guide to query correction, but to obtain the statistics of edits giving rise to errors. The method of identifying such pairs depends not only on the same IP address and near concurrency in time but also on a measure of closeness between query words. For this purpose we use an edit distance of one, two, or at most three edits. We also insist that if there is more than one edit, then the different edits occur separated by at least one character so that a proper context for each edit can be determined and the edit itself is not in question. We justify this based on the original observation of Damerau (1964) that 80% of spelling errors are produced by a single edit (deletion, insertion, replacement, or transposition). We make no claim that more complex edit operations do not occur, but we attempt to approximate them by combinations of single edits.

Our claim that the data we have collected represents spelling errors is supported by the fact that if one finds a term in the query logs that is not in the PubMed database and one looks at terms coming from queries by the same user preceding or following in time and close in lexical space, one is much more likely to find such terms following than preceding in time. This is shown by Figure 1 where it is evident that given a query term not in the PubMed database, one is much more likely to find a potential correction occurring after the term than before it. We believe the only reasonable explanation for this observation is that this asymmetry indicates people are constantly performing corrections on erroneous queries in order to obtain hits in the database. The fact that some correct terms appear before their erroneous counterparts we attribute to the fact that people not uncommonly type a term correctly and then have to repeat it and can make a typographical error on the second attempt that was not present on the first typing. Nevertheless we prefer to trust a correction that follows a query term. Figure 1 is completely based on query words that are not found in the PubMed database. However, there is also good evidence that people do not correct only terms that are not in PubMed, but also correct terms which are simply seen at low frequency in PubMed data. This is shown in Figure 2. Here we see that over the frequency range from 1 to 100 query terms are at least an order of magnitude more likely to be followed by a high frequency lexically close term than to be preceded by such a term. Again the asymmetry argues for the relatedness of such query terms pairs and that the second term in the pair is present as a correction for the first term.

Our data is the result of collecting such edits as we have described over 63 days worth of PubMed log files. We have collected approximately 1 million edits as summarized in Table 1. All multi-edit terms were required to have at least four times as many characters as edits in order to ensure that edits would truly be corrections. This is in addition to the condition that corrections have at least ten times the frequency in PubMed as the terms they are assumed to correct.

The data collected was collected with a single letter of context on either side of the edit. Both the beginning and the ending of a word were marked with special characters so that they could also function as context and make the correction process specific for the beginning and ending of words, respectively. One will notice that our data suggests that 87% of all misspelled words are the result of a single edit error. This is somewhat higher than the figure of 80% observed by Damerau (1964) but is consistent with our requirement that multiple errant edits occur with a letter of context separating them. This naturally reduces the number of higher order mistakes seen.

BASIC ASSUMPTIONS OF THE METHOD

In order to evaluate the expression (1) we must not only have information about the likelihood of edits. We must also be able to make estimates of the prior probabilities $P(w)$. These are the probabilities that the various words appearing in the PubMed database will be intended as query terms by users. We have examined the terms occurring in the PubMed database and we have found that they are used as query terms in direct proportion to their frequencies in the database. This is shown in Figure 3 where the straight line indicates a direct proportionality. The line is somewhat noisy at high frequencies due to sparse data and it makes a small bend at low frequencies indicating that at the very lowest frequencies fewer terms are used in queries. We would expect this bend due to the fact that the millions of very low frequency terms tend to be unknown to most users. Thus we can use the database frequency of a term as a surrogate for the probability that that term would be intended as a query term input by a user provided we discount the value at low frequencies. In fact our discounting at low frequencies is sharper than the bend in the curve of Figure 3 because at those low frequencies much of what users input is a misspelling and not what they intended. We discount by the formula

$$f' = f * 10^{0.075*(f-80)}, \quad f < 80 \qquad (2)$$

where f is the original database frequency and f' the discounted frequency. Thus our first basic assumption is that we can let the database frequency of a term stand in the place of $P(w)$ in (1) provided we apply the discounting given by (2).

Our second basic assumption is that people make spelling errors at a higher rate when composing queries than they do when composing text for the PubMed database. This is supported by the data quoted in the introduction on the rate of spelling errors in search engine queries (as high as 26%) when compared with data on spelling error rates in printed text of less than 5% (Kukich 1992). Printed text such as appears in PubMed is generally subject to an editorial process and automatic spell checking will have been applied as well in many cases. Further, printed text is often the product of more than one author's efforts and may be expected to have less spelling errors for that reason. Thus we believe our assumption is not unreasonable.

We use this assumption to decide when to correct a word that already appears in the database. Suppose *s* is a word that appears in the database and *w* is the word determined by evaluating the expression (1) for *s*. Then to decide whether we should suggest *w* as a correction for *s* we ask whether the inequality

$$P(s) < P(s \mid w)P(w) \tag{3}$$

is satisfied. If *s* is primarily a misspelling of *w*, we might expect an equality in (3) provided $P(s|w)$ is estimated based on the error rates prevailing in the PubMed database. In that case when we estimate $P(s|w)$ based on the higher error rates gleaned from the user logs we expect the inequality (3) to be satisfied. If it is we take this as some evidence that *s* is likely a misspelling of *w*. Of course just the inequality (3) alone gives a justification for suggesting *w* as a correction for *s* because the left side of the inequality is the probability that the user would intend *s* as the query term and the right side is the probability that the user would have intended *w* as the query term, but by introducing errors would have produced *s*. In the actual application of (3) we substitute the database frequencies of *s* and *w* for the probabilities of $P(s)$ and $P(w)$ and we use the discounting in (2) where appropriate.

In order to apply the formulas (1) and (3) we must also estimate the probability, $P(s|w)$, that while trying to produce *w*, errors are introduced that actually produce *s* We generally follow the “maximum probability alignment” or “minimum edit distance” method as described in Jurafsky and Martin (2000). We estimate $P(s|w)$ as the product of the probabilities of a sequence of edits which will produce *s* from *w*. Since there is often more than one such sequence we take the sequence that yields the highest probability as our estimate for $P(s|w)$.

THE ALGORITHM: BASIC FUNCTIONS

Here we begin the description of the algorithm by describing how spelling correction is handled at the most basic level. Our objective is to offer a correction only if we can do so with an assurance of being correct in our offering at least 70% of the time. This requirement influences to some extent how the basic functions are constructed. Assume *s* is a string to be corrected.

OneEdit

We estimate $P(s)$ and $P(s|w)P(w)$ over all the *w* in the database that are within one edit of *s*. This is done by using database frequencies (with discounting as appropriate) and edit probabilities and then normalizing the resulting estimated figures to sum to one. Let *c* denote the term with the largest estimated probability and let P_c denote that probability. If either $P_c > 0.7$ or $P(s) < 0.05$ accept *c* as a correction. Otherwise offer no correction. The justification for offering *c* as the correction when $P(s) < 0.05$ is that in that case we can reject at the 5% level the hypothesis that *s* is what the user intended and we had as well give our best guess as a correction. Thus our strategy is to offer a correction if we are quite certain we are correct and also when we are quite certain the input string was not intended, even though in the latter case we may be much less certain that the correction is the right one.

TwoEdit

We estimate $P(s|w)P(w)$ over all *w* in the database that are two edits away from *s*. If there are such strings we return the most probable one as the accepted correction. Otherwise no correction is offered.

RecursiveEdit

If we tried to produce a correction with two edits and failed, we would have produced an alignment of an initial segment of *s* with an initial segment of a word *w* in the database involving two edits. We can rate such attempts by how many of the letters in *s* they use. We let *m* denote the maximum rating obtained by any such partial alignment. We then ask for that partial alignment that gets the rating *m* and also has the highest probability among all such partial alignments rated *m*. We call this the best partial alignment. We can then repeat this procedure each time beginning with the best partial alignment produced on the previous iteration. If we require the algorithm to make some advancement along the string *s* at each stage and discontinue the process if it fails at any point, we then obtain an algorithm that will either produce a complete alignment or end without producing any alignment with only a few iterations. If the algorithm does terminate with a suggestion we require that the result pass a test of similarity to *s* which we term a sanity check (see below). If it does it is accepted as a correction. Otherwise no correction is offered.

StringSplit

We attempt to introduce a space at some point in the string to convert it to two words. If the resulting words are both found in the database it becomes a candidate split for the string with a rating equal to the lesser of the database frequencies of the two words produced by the split. If there is a split the highest rated split may be offered as a correction. It is generally required to have a rating above some lower limit in order to be accepted. If it does the split is offered as a correction. Otherwise no correction is offered.

If a word such as “phosphatase” is divided by the erroneous introduction of a space internally as in “phosp hatase” this can be corrected by a single edit operation which deletes the extra space. Thus no special mechanism for correction is required beyond the OneEdit, TwoEdit, or RecursiveEdit functions described above. However, if two words are accidently run together as in “venombite”, then edit operations may not suffice because the string “venom bite” does not occur among the strings being searched for a correction. It is for this reason that StringSplit is needed.

In addition to the basic functions just given we also use two types of checks to be sure a string has not been altered too much in the correction process. We call these sanity checks.

Sanity1

This check compares the first three characters of *s* and a putative correction. If in comparing the characters at position 0, the characters at position 1, and the characters at position 2 there is at most one difference between the strings then the correction passes this test.

Sanity2

This test is a more extensive test in which a point is counted if a character is replaced, a point is counted if one or two characters in a row are either inserted or deleted, but transpositions are assigned zero cost. One then compares a string s and a putative correction by comparing the first word in each, the second word in each, etc. The test is passed if in any such comparison one sees a cost of transformation that does not exceed two points for each pair of words compared.

Estimating regions of success

While the edit functions OneEdit, TwoEdit, and RecursiveEdit suffice to make corrections in strings, they are not equally successful on strings of any length. As a rule the shorter the string the more difficult it is to correct. There are two reasons for this. First, a shorter string has less useful context surrounding the errors by which to identify the intended string. Second, the space of all strings is much more densely populated in the region of shorter strings (Kukich 1992). This problem of a densely populated space is clearly an issue in the PubMed database where there are very many strings arising as abbreviations. Because of this issue we examined the performance of the edit functions by a simulation. Single words were sampled randomly from the database with a probability proportional to their frequency in the database. Once a word was sampled either one, two, or three edits were introduced randomly into the word using the context specific edit probabilities that we collected from the user logs. An attempt was then made to correct the spelling error using the edit functions. We compiled the data into tables giving the detailed results for the different numbers of edits and specific to the length of the string that the algorithms were given to correct. The results are contained in Table 2-Table 4. The data in Table 2 shows that it is very difficult to correct really short strings. Based on this data we do not attempt to correct single words of length less than five or six. Likewise Table 3 suggests one needs a string of about length nine to reliably correct two edits and in the same way Table 4 suggests a string of approximately length twelve is needed to reliably correct three edits. Similar data can be simulated for two word phrases. We have made use of such data in constructing our algorithm.

THE ALGORITHM

A trie is a tree structure which can hold many individual strings and given any new string the trie allows for very efficient testing to see whether this new string is one of those stored in the trie (Sedgewick 1998). In order to efficiently search for the best correction for a query string *s* as required in expression (1), we use a trie structure (Kukich 1992; Brill and Moore 2000). All the database terms to be searched are loaded into this trie. Then, as pointed out by Hall and Dowling (1980), there are two basic approaches available. One can generate all the strings which are close (within say one or two edits) of the string *s* and see which are in the trie. Or one can attempt to search the trie directly with the string *s* making corrections as needed to produce a match. The problem with generating all the strings that are close to *s* in edit space is that one will generate many nonsense strings that are of no interest and must then search each of these to see if it is in the database. We prefer the direct search of the trie for its efficiency. For example, if one traces a match of the first *k* letters of *s* into the trie and cannot extend this match to the $k + 1$ st letter, then one can conclude there must be an error in the first $k + 1$ characters of *s*. Further one does not have to examine all possible edits, but only those that will extend a match into the trie. This results in a significant savings in time without overlooking any possible matching string in the trie.

Because we must correct errors in phrases of variable length we actually use three different tries in the algorithm. First, we construct a trie, Tr123, of all the phrases consisting of one, two, or three tokens that are recognized by the search engine. If the query string *s* consists of one or two tokens we search for a correction in Tr123. This allows a correction to have more or less tokens than the query. For example the query “apop tosis”, mistakenly broken into two tokens, will yield the correction “apoptosis” consisting of one token and the query “bcell lymphoma”, mistakenly run together to form two tokens, will yield the three token correction “b cell lymphoma”. If the query string *s* consists of three or more tokens we search for the phrase, *s*′, consisting of its first two tokens in a trie, Tr2p. Tr2p is a trie containing all the one or two token phrases that are the initial one or two tokens of phrases of three or more tokens and that are recognized by the search engine. If we find a match, even a correction, we attempt to extend this correction in the trie, Tr3+, which consists of all the phrases consisting of three or more tokens that are recognized by the search engine. For example the system makes no correction to the query “doman” (a persons name), but given the query “dna binding doman” it first checks that “dna binding” occurs in Tr2p and then extends this to the correction “dna binding domain” in Tr3+. In this way we avoid attempting a very long match, which would be costly in time, unless we have some evidence that a long match is possible based on the initial part of *s*. If the initial match of *s*′ into Tr2p fails we seek a match of *s*′ into Tr123, etc. Thus the algorithm is organized around the number of tokens contained in the search string *s*.

We proceed to give pseudocode for the different cases or numbers of tokens in a string. In what follows we will let $l(s)$ denote the length in characters and $f(s)$ denote the database frequency for any string s. Any string not in the search engine vocabulary is counted as having a database frequency of zero. Note that we use the word RETURN to signal the end of a calculation with the module either giving back a suggested correction or not, but in either case all lines following the RETURN to the end of the module are ignored. We have also used the word “Stage” to mark different parts of the algorithm for readability and there is some correlation in the expense of the calculations with higher stage numbers correlating with the more expensive computations.

```
SingleTokenModule {
Stage 1
IF $l(s) < 5$ THEN RETURN without a correction.
IF $f(s) > 1000$ THEN RETURN without a correction.
ELSE CALL OneEdit for $s$.
    IF $s$ is not within one edit of any string in the database THEN set $R$=0.
    ELSE IF OneEdit finds a correction $c$ THEN set $R$=1.
    ELSE set $R$=2.
Stage 2
IF $R = 0$ and $l(s) \geq 9$ THEN
    CALL StringSplit
        If StringSplit produces two strings $c$ and $c'$ with $f(c) \geq 500$ and $f(c') \geq 500$ THEN
        RETURN $c$ and $c'$ as the correction.
    CALL TwoEdit for $s$.
        IF TwoEdit produces a correction THEN RETURN this as the correction.
ELSE IF $R$=1 and $l(c) \geq 5$ THEN CALL OneEdit for $c$.
    IF OneEdit succeeds change $c$ to represent the new string it produces (and leave $R$=1).
Stage 3
IF $R = 1$
    IF $l(s) \geq 9$ and $f(c) < 80$ THEN CALL TwoEdit for $c$.
        IF TwoEdit gives a correction $c'$ and $f(c')$>80 and $f(c')$>10 $f(c)$and Sanity1 is passed
        THEN RETURN $c'$ as the correction.
    RETURN $c$ as the correction.
IF $R = 2$
    IF $l(s) \geq 9$ and $f(s) < 80$ THEN CALL TwoEdit for $s$
        IF TwoEdit gives a correction $c$ and $f(c)$ >80 and $f(c)$ >10 $f(s)$ and Sanity1 is passed
        THEN RETURN $c$ as the correction.
    RETURN without a correction.
Stage 4
IF $l(s) \geq 12$ THEN CALL RecursiveEdit for $s$.
    IF RecursiveEdit gives a correction $c$ and it passes Sanity2
        THEN RETURN $c$ as the correction.
CALL StringSplit.
    IF StringSplit gives a split $c$ and $c'$ with $f(c) > 0$ and $f(c') > 0$
        THEN RETURN $c$ and $c'$ as the correction.
RETURN without a correction.
}
```

As an example suppose the query string is “ribonflaven”. Then because this string has a length greater than 5 and does not occur in the database the SingleTokenModule will attempt a correction. In Stage 1, OneEdit is called and produces the correction “ribonflavin” which occurs 1 time in the database. In Stage 2 another OneEdit is tried on the correction and produces “riboflavin” which occurs 7380 times in the database. In Stage 3, because of its high frequency in the database, “riboflavin” is returned as the correction. This example illustrates two of the guiding principles in designing the spell checking algorithm. First, small changes in a query string are always preferred over large changes. Second, changes that produce a word found in the data are always more believable than changes of comparable magnitude that do not. Here one edit gets us from the string “ribonflaven” to the string “ribonflavin” which appears in the data and so has the plausibility of being at worst a misspelling of something in the database. Then one more edit gets “ribonflavin” to the high frequency string “riboflavin”. This chain of two small changes has more evidence in its support than simply asking for the results of TwoEdit. In general decisions are made based on the plausibility of the results where we gauge plausibility by:

A. Smaller changes are more plausible.

B. Changes that produce a string in the database are more plausible than changes of the same magnitude that do not.

C. Changes that produce a string of high frequency in the database are preferred over those of similar magnitude that do not.

The SingleTokenModule follows these principles as it descends through stages looking for the most plausible solution first, but trying successively less plausible methods until either a solution is found or the attempt fails to produce a correction. All the different methods of correction are present because we actually found them necessary in certain cases. There are certain constants in SingleTokenModule (and in the other modules) that were chosen because they gave reasonable results in trials. They were chosen empirically and no formal evaluation has been done. We will return to this issue below.

All the searching in SingleTokenModule is done in the Tr123 trie. The same is true for the TwoTokenModule we are about to describe. When we are given a two word query a new element is introduced into the problem. That is the issue of context. It is possible that one of the words is correct and can be used as context to more effectively correct the other. On the other hand the two words need not be closely related as would occur in a meaningful phrase. Thus we must have a strategy which tells us when to attempt to use context and when to avoid it. This strategy is an important part of the overall plan for multi-word correction. In the following pseudocode we will let the two token query be denoted by s_t where s and t are the individual tokens. In what follows we will make use of the basic edit functions defined in the previous section. However there are some constraints that we have found useful that apply to the editing of s and t regardless of $l(s\_t)$.

Constraint1

If the length of a token is less then three do not edit it. Assume it is correct.

Constraint2

If the length of a token is less then seven make at most one edit in it.

These constraints apply to a given token regardless of the length of the other token in the phrase. We believe that tokens of one or two characters are unlikely to be misspelled and we use them as fixed points by which to guide the correction process.

```
TwoTokenModule {

Stage 1

IF $l(s\_t)<7$ THEN RETURN without correction.

Set $f_m = \min(f(s), f(t))$.

IF $f(s\_t)>5$ and $f_m>500$ THEN RETURN without correction.

IF $f(s\_t)>0$ and $f_m>50$ and either $l(s) \leq 4$ or $l(t) \leq 4$
    THEN RETURN without a correction.
Stage 2

CALL OneEdit for $s\_t$ and set $R=0$.
    IF OneEdit gives a correction $c$ THEN set $R=1$.
IF $R=1$ THEN CALL OneEdit for $c$.
    IF OneEdit gives a correction $c'$ THEN set $c = c'$
IF $R=0$ THEN CALL TwoEdit for $s\_t$.
    IF TwoEdit gives a correction $c$ THEN set $R=1$.
IF $R=0$ and $f_m \geq 100$ THEN CALL SingleTokenModule for each of $s$ and $t$ separately and
RETURN the result.

ELSE IF $R=1$ and $f(c) \geq f_m$ THEN RETURN $c$ as the correction.

Stage 3

CALL StringSplit for $s\_t$
    IF StringSplit produces strings $s'$ and $t'$ with $f(s')>500$ and $f(t')>500$ THEN
        RETURN $s'$ and $t'$ as the correction.
IF $l(s\_t)>20$ and either $f_m = 0$ or both $l(s) \geq 7$ and $l(t) \geq 7$ THEN
    CALL RecursiveEdit for $s\_t$
    IF RecursiveEdit gives a correction $c$ that passes Sanity2 THEN
```

```
        RETURN $c$ as the correction.
    CALL SingleTokenModule for each of $s$ and $t$ separately and RETURN the result.
}
```

As an example of the action of TwoTokenModule consider the query string “gammg globulin”. This string only occurs 1 time in the database and because “gammg” only occurs 2 times in the database “gammg globulin” gets by Stage 1 in the processing and is a candidate for correction. OneEdit produces the correction “gamma globulin” and a repeat call to OneEdit produces no improvement so that this is accepted as the final correction. Because the string went from a frequency of 2 by a single edit to a final frequency of 15,568 the correction has high plausibility. Now consider the query string “academic attitude”. This phrase does not occur in the database, so it passes through Stage 1 and becomes a candidate for correction as a phrase. However, the only correction that is found is the string “academic aptitude”, which occurs 30 times in the database. Because of the low frequency of this phrase it is not accepted as a correction. We take frequency as a measure of plausibility and “academic” occurs 52,629 times and “attitude” 144,536 times in the database. We will state this as one final principle of plausibility in making corrections.

D) Most errors in typing phrases produce as least one word that is of lower frequency than the phrase that was intended. Therefore a phrase correction is only plausible when it is of higher frequency than at least one of the words in the initial query string.

The pseudocode for SingleTokenModule and TwoTokenModule gives a detailed view of how we handle one and two token strings. Finally, we shall give a somewhat abbreviated description of how we handle strings with three or more tokens. Let *s_t_u* denote such a string where *u* is allowed to stand for possibly more than one token. We follow a number of steps:

I. We see if *s_t* occurs in Tr2p. If it does not we search for a correction for *s_t* in Tr2p. The search is the same as the search in TwoTokenModule except at stage three we only allow the RecursiveEdit as an option and we only require $l(s\_t)>20$ to apply it. StringSplit and applying the SingleTokenModule twice are not options at this point because their success would prematurely rule out other options that are preferred. The weaker condition for application of RecursiveEdit is used because the result will not be final until a longer match (with more context) is obtained.

II. If in I we find *s_t* or a correction for *s_t* in Tr2p then we attempt to extend this initial match to a match of *s_t_u* in Tr3+. For this extension we use a form of the RecursiveEdit. If this produces a match which passes Sanity2 we accept this as the correction for *s_t_u* and are done. If it does not produce a match we attempt to back down the solution trie to find a match into Tr3+ that does not involve all of *s_ t_u* and passes Sanity2. If this can be done an alignment of the correction is done with the original query string to determine what part of the string remains to be corrected. We then accept the partial correction and call the process recursively to correct the remaining string.

III. If I finds a match or correction *c* in Tr2p, but II fails to yield a match into Tr3+ and if *c* consists of a single token we attempt to extend this to a match in Tr123. If this succeeds and the match passes Sanity2 we accept it as a correction and again must deal with finding any remaining string to match just as in II.

IV. If not even a partial solution is reached in I-III, then we attempt to find a correction for *s_t* in Tr123. This is done by basically applying TwoTokenModule, again with minor modifications. The modifications are two. First, in stage 2 with a high frequency limit and then again in stage 3 with a lower limit an attempt is made with StringSplit to split *s_t*. If this is successful the first part of the split is accepted as a part of the correction and the process is called recursively on the second part of the split and any remaining tokens beyond. Second, if all else fails then SingleTokenModule is called for *s* only and the result is accepted as a partial correction (or possibly no correction) and the process is called recursively to correct *t _ u*.

With an example we illustrate the importance of context when there are three or more tokens. Consider the query string “amytrophic laterl slersos”. The first two tokens are first corrected to “amyotrophic lateral” and then the system attempts to extend this by correcting “slersos”. This last string “slersos” has only seven characters and three errors would ordinarily make it difficult to correct, but here there are few strings that begin with “amyotrophic lateral” other than the correct one and so the system easily corrects “slersos” to “sclerosis”. Because of initial tokens which provide context we are able to relax the constraints (Constraint1 and Constraint2) in the extension process.

As a final comment on the construction of the algorithm we note that in OneTokenModule and TwoTokenModule and less visibly in the processing of query strings of three or more tokens, there are a number of parameters. These parameters were chosen empirically by observing the functioning of the algorithm on querys coming in to the PubMed search engine and making adjustments. We make no claims that the choices incorporated here are optimal. In fact one of the difficult issues is to determine what optimal should mean in a setting of this kind. One might assume a criterion of maximizing the number of reasonable spelling suggestions made. On the other hand the ultimate goal is to please the users and optimally facilitate their search. From that point of view there is some cost for making incorrect or even ridiculous suggestions. If users do not have faith in the reasonableness of the suggestions they may be less inclined to use them. Our approach has been somewhat conservative in trying to avoid high risk suggestions and to achieve a high accuracy rate rather than an overall maximum number of reasonable suggestions with a lower accuracy rate. In other words we have been more concerned about precision than recall.

This completes our description of the algorithm.

CLEANING UP THE PUBMED DATA

Generally the misspelled words in the PubMed database are low frequency and it is this property that allows spell correction to operate based on the vocabulary of the database. However, there are some terms that are misspelled or at least not optimal as query terms that are relatively high frequency in PubMed. Because of this we undertook to try to deal with this problem. We examined all the one and two word phrases that occurred in at least a threshold number of documents in PubMed and that also were one edit from another term in PubMed that had a database frequency at least ten times as great. We made the assumption that if two such terms had a significant tendency to occur in the same context then the lower frequency member of the pair was a misspelling or at least a non-optimal version of the higher frequency term. For single token terms we used a low frequency threshold of 20. For two token terms, which are less common, we used a low frequency threshold of 9. The important consideration in choosing the low frequency threshold is to simply have enough data to allow the computation of a reliable statistic. We found it useful to handle the single token and the two token cases somewhat differently also in how they were tested.

Single token

Assume that a pair of terms one edit apart are represented by T_1 and T_2. Then we apply a test based on the hypergeometric distribution (Larson 1982). The situation is illustrated in Figure 4. We compute the *p*-value that the two terms would co-occur in the number of documents they are observed to co-occur in or more, if the two terms were only random in their relationship to each other. We found 62,720 pairs in the database that satisfied the frequency requirements and were one edit apart. When the hypergeometric test just described was applied the result was 10,922 single token pairs that were related with a *p*- value less then 0.01. This means that we can expect 99% of these term pairs to be significantly related. A sample of such pairs is shown in Table 5. In the majority of cases the low frequency member of a pair is a misspelling. In some cases it is simply a non-optimal query term because there is a much higher frequency term with essentially the same meaning for search purposes.

Two tokens

In this case we found 11,762 pairs of two token phrases that satisfied the frequency requirements. We first applied the hypergeometric significance test just as for the single token case. This resulted in the identification of 1,836 pairs that were significantly related. If the hypergeometric test failed to indicate significance at the 0.01 level, we then applied a more stringent test. The constructions involved are depicted in Figure 5 where we have depicted the case of the pair of phrases “myocardial infraction” and “myocardial infarction”. These two phrases differ only in their second words and we have used the common first word “myocardial” to define the context or set of documents of interest.

Within this set the set of documents containing the misspelling “myocardial infraction” corresponds to the rectangle and the set containing the correct phrase “myocardial infarction” corresponds to the ellipse. We randomly sampled three sets: *B1* from documents containing the misspelled phrase, *G* from documents that contain the correct phrase but not the misspelled phrase, and *B2* from documents containing neither phrase but containing the word “myocardial”. Each of these sets consisted of a thousand randomly selected documents if that many fell into the category being sampled. If the set of documents available was smaller than one thousand the whole set was taken as the sample. The sampling was used to place a limit on the amount of computation necessary for any phrase pair to be evaluated. We then applied Naïve Bayesian learning to learn the difference between *G* and $B1 \cup B2$. With the weights obtained in this way we scored all the documents in $B1 \cup B2$ and arranged them in order of decreasing score. We then applied the Wilcoxon-Mann-Whitney test to see if the sum of the ranks of the members of *B1* was less than would be expected. This would mean the members of *B1* were higher scoring than expected or put another way were more similar to the members of *G* than were the members of *B2*. We applied this test to the 9,926 pairs that remained after the 1,836 pairs found by the hypergeometric test were removed. As a result we identified an additional 5,628 phrase pairs that were significant at the 0.01 level. A sample of the phrase pairs found in this way is given in Table 6.

The 10,922 single token pairs and the 7,464 two token pairs are not removed completely from consideration. Rather they have their frequencies reduced to one for the purposes of the computations involving expressions (1) and (2). Thus they are much more likely not to be chosen as a correction for a query. However, they remain as possible intermediate steps in a sequence of operations leading to a correction. If they do appear as such an intermediate the chances have been enhanced that the final correction will be the high frequency term they were found to be related to in the statistical testing just described.

One may ask why we did not use the WMW test for the single token phrase pairs. The reason is that we found many false positives when we tried to use it. Our attempt involved a picture similar to Figure 5. However, we had no contextual word like the word “myocardial” in that picture to focus the computation. We therefore sampled B2 from the whole remainder of PubMed data outside those documents that included one of the tokens of interest. Then if the low frequency token in the pair was not a misspelling the sample B1 would be from a meaningful topic that was quite unrelated to G. The result was the documents in B1 could be quite consistently more related or less related to G than the general random sample B2. If more related the statistical test could be easily satisfied at the 0.01 level and still there not really be a meaningful relationship between *B1* and *G*. Thus we abandoned the effort. Perhaps some refinement of the test could be used in this way. If so, it could prove quite useful because one cannot expect the hypergeometric test to work in all important cases. This is true because when a misspelling does appear in a document it may be a consistent error and the correctly spelled term may not appear. In such cases a context based test like the WMW test we used has a much better chance to detect the error.

PERFORMANCE ISSUES

For the PubMed database the tries used in the spelling correction algorithm currently involve 14,267,366 one, two, and three token strings in Tr123; 2,775,111 strings of three or more tokens in Tr3+; and 1,772,383 initial segments of strings from Tr3+ in Tr2p. On a normal work day the PubMed query engine receives roughly 3 million user queries and this generates well over 3 million queries to the spell checking algorithm. This is because many queries are complex and involve parsing of punctuation and Boolean operators with the result that multiple fragments are produced and checked for spelling. The spell checking algorithm actually suggests corrections for about 10% of user queries, but any suggestion that is produced is checked to see if it posts (if it retrieves some documents in the database). Any correction that does not post is ignored. The result is that a suggested correction is made to the user on about 7% of user queries. When we first began making suggestions to users, they were accepted by the user 36% of the time. After approximately six months users were accepting suggestions at a 40% rate. Now after approximately a year since deployment, on the most recent Monday there were 3,275,624 queries to the PubMed search engine and 243,853 PubMed spell suggestions were made to 80,785 unique IPs and 109,526 (45%) of spell suggestions were clicked from 45,285 unique IPs.

A small set of user queries, 1,323, were examined and 110 of these had suggestions made by the spell checking algorithm. Of the 110 corrections offered 96 were judged by two judges (consulting together) to be good and 14 bad. This is an 87% success rate with a 95% confidence interval of (81%,92%). This is well above the target figure of 70% correct at which we had aimed and we believe part of the reason for this is that suggested corrections that do not post are ignored by the system.

Currently the spell checking algorithm is running on six Dual Intel Xeon 3.6 GHz machines each having 6 GB of RAM. It is written in C++ and is running under linux in 64 bit mode. Its use has added, on average, about 25% to the response time of the PubMed query engine, but as a practical matter it adds very little to the response time of correctly spelled queries. The spell checking algorithm is implemented on six servers because it is now being used for query correction on fourteen different NCBI databases of which PubMed is simply the largest.

DISCUSSION

Table 7 gives examples of corrections that the algorithm is capable of making. These examples are chosen because they illustrate the effect of context and some of the extremes of pathology, not because the misspellings are typical. Of course not all the suggested corrections fair so well and it is of some interest to see the kinds of errors that are made. We examined just a little over 500 suggestions made by the spell checker that were not accepted by users and found what we thought were the most glaring mistakes. They are contained in Table 8.

One may note that five of the seven occur in phrases in which two or more edits have been attempted. The fact that “Sapna Baht”, a persons name, is two edits from the phrase “sauna bath" is just a coincidence that is not common. The problem with "periostin" is a result of the phrase "periods in" which should not have been accepted into the search engine's vocabulary.

The problems with "Daniel K E" and "stem cell ros" are a consequence that we do not enforce Constraint1 and Constraint2, respectively, when the phrase has more than two tokens. The algorithm might benefit in terms of accuracy if we did, but would be more complicated. For both "bisexual molest" and "pancreas & transplation" the available context is not used. This is because neither corrects to a phrase in the system. Rather "transplation" and "molest" are corrected in isolation. Of course "molest" is correctly spelled, but it only occurs 23 times in PubMed documents while "modest" occurs over 28 thousand times. One can see that "molest" is more reasonable than "modest" because of the other part of the query, however, currently the system only uses context if it is part of a valid phrase in the system. Finally, there is the case of "cupper hair". Here the word "cupper" occurs ten times in PubMed (at the time of this writing and not counting the author field). Once it is a person's name and the other nine times it is a misspelling of "copper". The algorithm would correct "cupper" to "copper" (over 53 thousand occurrences) except it prefers phrase corrections where context can more effectively guide the process. This time, however, it produces a mistake. The user may well have been interested in Menke's disease which is caused by intestinal malabsorption of copper and is characterized by kinky hair (which is colorless). Unfortunately "copper hair" is not a characteristic of Menke's disease nor does the phrase even occur in the PubMed database.

One may ask how our spelling correction accuracy compares with that of others who have used the noisy channel model. Church and Gale (1991) quote an accuracy of 87% in correction of a set of 332 misspellings identified by the Unix *spell* utility and whose correction was agreed upon by at least two of three human judges. All of these misspellings were characterized by having exactly two possible one edit corrections in a word list compiled by the investigators from standard sources. When the spelling correction model was augmented with contextual information through a language model they obtained an improvement to 89.5%. Here we can say that our accuracy figure is similar to theirs though there are many questions regarding how comparable the testing is. First, we do not use a language model but something less, though context is not completely ignored in our process. Second, they limited their process to single edits whereas we allowed multiple edits. Finally, they limited their testing to correction where there were only two options as answers and this would seem to enhance their accuracy. Thus it is difficult to draw conclusions from such a comparison.

A second version of the noisy channel model for spelling correction has been put forward by Brill and Moore (2000). They employ a more sophisticated model of edits in which a single edit can make a multi-character correction. They also invoke a wider context than the single character preceding a correction used by Church and Gale or the single character on either side which we use. They studied a 10,000 word corpus of common English spelling errors paired with their correct spellings. They trained on a subset of 8,000 of these and tested their system on the remaining 2,000. In the testing process they used a dictionary of 200,000 entries which included all words in the test set. They found an accuracy of 95% without a language model. In order to evaluate the effect of a language model they computed corrections for the same test words as they occurred in context in the Brown Corpus. This led to an accuracy figure of 95% and a corresponding figure of 93.9% without the language model (because results are computed per token instead of per type). Since our correction accuracy is computed per token also, it is these latter figures that are most comparable. They used a context of 3 characters on either side of an edit as context to obtain this result. Here their performance figures are better than ours. But one must ask how dealing with only the most common errors in English would affect their performance. Comparably, we deal with the full range of errors that occur possibly involving multiple tokens, though the most common errors will have the biggest effect on our accuracy. Another factor involved here is the size of the dictionary used in the correction process. In our case the number of unique tokens is over 2.5 million, while Brill and Moore use a word list of 200,000. Thus our dictionary is more than an order of magnitude larger than theirs. As long as the dictionary contains the correct answers, the smaller the dictionary the easier the correction process. The fewer the number of correct answers the less dense they are packed (Kukich 1992) and the less likelihood that different dictionary entries will compete to provide the correction for a misspelled string.

Because of the differences in how context can be used in a search engine query as opposed to natural language text and because of the differences in the size of the dictionary it is not easy to draw clear conclusions from these comparisons. One thing that does seem of interest is the wider context within a string that Brill and Moore use to condition an edit. They found an improvement of about 2% in accuracy in using a window of three characters on either side of an edit instead of a window of only one character on either side. This suggests that we might see a similar improvement if our algorithm used a wider context. What we do not know is how such a change would affect the speed of the algorithm. This issue bears further investigation.

Another possible avenue for improvement of the algorithm is some form of phonetic correction. It is recognized that most misspellings (approximately 80%) are single edit errors where an edit is understood in Damerau’s sense (Damerau 1964) of an insertion of a letter, a deletion of a letter, a replacement of a letter, or a transposition of two adjacent letters. However, phonetic errors often involve more letters and are more difficult to correct (Kukich 1992). Zobel and Dart (1995) compared Soundex and Phonix (Gadd 1990) with edit distance based methods and concluded that the phonetic based methods were inferior to the edit distance approach in finding good matches for strings in a large lexicon. We examined the Metaphone (Philips 1990) algorithm and attempted to use it to correct misspellings in simulations where errors were generated as in Table 2–Table 4. In all cases we found the results inferior to what we were able to produce using the noisy channel model and expressions (1) and (2). In our experience phonetic correction works well in some cases, but in others it identifies strings as similar that should not be identified or fails to make such an identification when we would want it to. For example Zobel and Dart point out that “mad” and “not” encode to the same string under Soundex and Phonix. Likewise we note that using Metaphone “phalanges” encodes to “flnjs” while “hpalanges” encodes to “hplnjs”. Thus a single edit error can become magnified under the encoding. Another question to ask in this setting is how many spelling mistakes occur in PubMed queries that are not correctable with one or two edits. This is relevant because our algorithm already works quite well on errors consisting of one or two edits. To examine this question we processed the same 63 days worth of PubMed user log files from which we obtained our edit probabilities and in a similar processing collected all the single token pairs where the first member of the pair was not within two edits of any string in the PubMed database but the second appeared in the database, while the two strings produced the same encoding under Metaphone (note we use the full encoding without truncation). We identified 5,781 such pair occurrences involving 2,894 unique pairs. If one optimistically assumed that one could correct the errant query string in all cases using the Metaphone encoding in this way this would at most yield 92 additional corrections a day to what we are already doing. Given that we commonly find users accepting over 90,000 corrections/day we are looking at an at most 0.1% increase in what users accept and more realistically probably less than half of that. Thus it is unclear whether phonetic correction is worth the overhead it would involve.

CONCLUSIONS

We have developed a spell checking algorithm that does quite accurate correction (≅87%) and handles one or two edits, and more edits if the string to be corrected is sufficiently long. It handles words that are fragmented or merged. Where queries consist of more than a single token the algorithm attempts to make use of the additional information as context to aid the correction process. The algorithm is implemented in the PubMed search engine and there it frequently makes over 200,000 suggestions in a day and about 45% of these suggestions are accepted by users. The algorithm is efficient in adding only about 25% to the average query response time for users and much of this is seen only for misspelled queries. There is the possibility of improving the algorithm by the use of more context around the sites of errors within words. There is also the possibility of improving the algorithm by learning how to make better use of the context supplied by queries consisting of multiple tokens. In both cases such an effort must consider how to maintain efficiency in the light of a huge vocabulary of phrases (>14 million) and individual words (>2.5 million) recognized by the search engine. There is also the possibility to use phonetic encodings to improve the handling of some of the errors that currently challenge the system. However, preliminary calculations suggest it would be difficult to make a major improvement by using phonetic encodings.

Acknowledgements

The authors would like to thank David Kenton and Pramod Paranthaman for insightful discussions and for their work evaluating the algorithm and Vladmir Sirotinin and Grisha Starchenko for their work incorporating the algorithm into search engine query processing. Thanks are also due to the anonymous referees for making helpful suggestions for improvements in the paper. This research was supported [in part] by the Intramural Research Program of the NIH, National Library of Medicine.

References

Beeferman, D.; Berger, A. Agglomerative clustering of a search enginne query log. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; Boston, MA, ACM Press. 2000.

Brill R, Moore RC. An improved error model for noisy channel spelling correction. ACL 2000. 2000

Church KW, Gale WA. Probability scoring for spelling correction. Statistics and Computing 1991;1:93–103.

Damerau FJ. A technique for computer detection and correction of spelling errors. Communications of the ACM 1964;7(3):171–176.

Gadd TN. PHONIX: The algorithm. Program: Automated Library and Information Systems 1990;24(4):363–366.

Hall PA, Dowling GR. Approximate string matching. Computing Surveys 1980;12(4):381–402.

Huang CK, Chien LF, et al. Relevant term suggestion in interactive web search based on contextual information in query session logs. Journal of the American Society for Information Science and Technology 2003;54(7):638–649.

Jurafsky, D.; Martin, JH. Speech and Language Processing. Upper Saddle River; New Jersey, Prentice Hall: 2000.

Kukich K. Techniques for automatically correcting words in text. ACM Computing Surveys 1992;24(4):377–439.

Larson, HJ. Introduction to Probability Theory and Statistical Inference. New York: John Wiley & Sons; 1982.

Leroy G, Lally AM, et al. The use of dynamic contexts to improve casual internet searching. ACM Transactions on Information Systems 2003;21(3):229–253.

McEntyre J, Lipman D. PubMed: bridging the information gap. Cmaj 2001;164(9):1317–9. [PubMed: 11341144]

Nordlie, R. “User revealment” - a comparison of initial queries and ensuing question development in online searching and in human reference interactions. SIGIR’99: 22nd International Conference on Research and Development in Information Retrieval, University of California; Berkeley, ACM Press. 1999.

Philips L. Hanging on the Metaphone. Computer Language 1990;7(12)

Sedgewick, R. Algorithms in C (Parts 1–4). Boston: Addison-Wesley; 1998.

Silverstein C, Henzinger M. Analysis of a very large web search engine query log. SIGIR Forum 1999;33 (1):6–12.

Spink A, Wolfram D, et al. Searching the web: the public and their queries. Journal of the American Society for Information Science and Technology 2001;52(3):226–234.

Survey. NPD Search and Portal Site Survey. 2000. Retrieved September 26, 2005, from http://www.searchenginewatch.com/sereport/article.php/2162791

Wang P, Berry MW, et al. Mining longitudinal web queries: trends and patterns. Journal of the American Society for Information Science and Technology 2003;54(8):743–758.

Wen JR, Nie JY, et al. Query clustering using user logs. ACM Transactions on Information Systems 2002;20(1):59–81.

Zobel J, Dart P. Finding approximate matches in large lexicons. Software-Practice and Experience 1995;25(3):331–345.

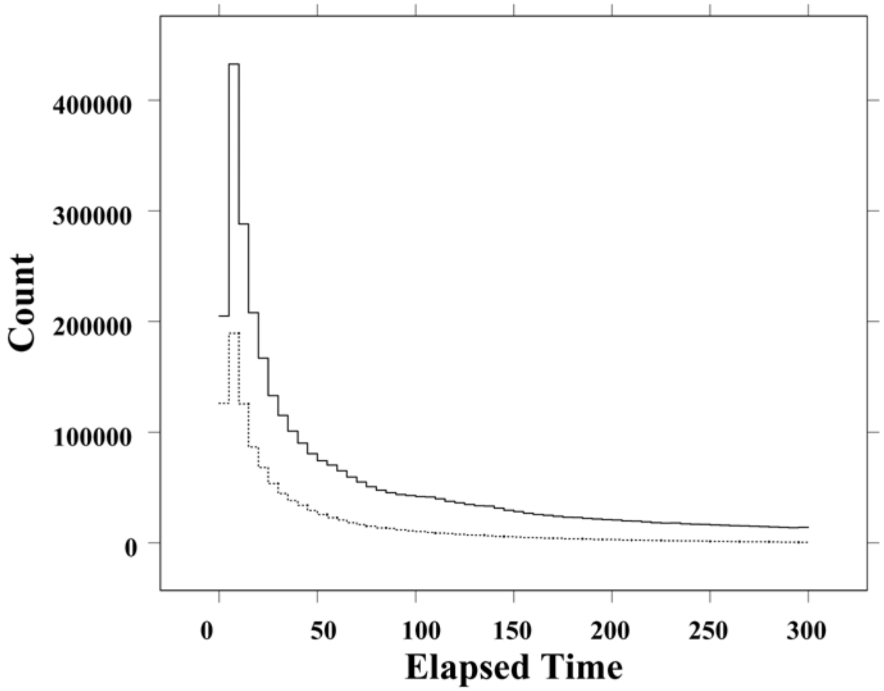

Figure 1.

The solid curve represents the number of query term pairs where the term contained in the PubMed database follows the term that was not in the database. The broken curve represents the same data when the term contained in PubMed precedes the term that was not in the database. In all cases the terms are within three edits of each other.

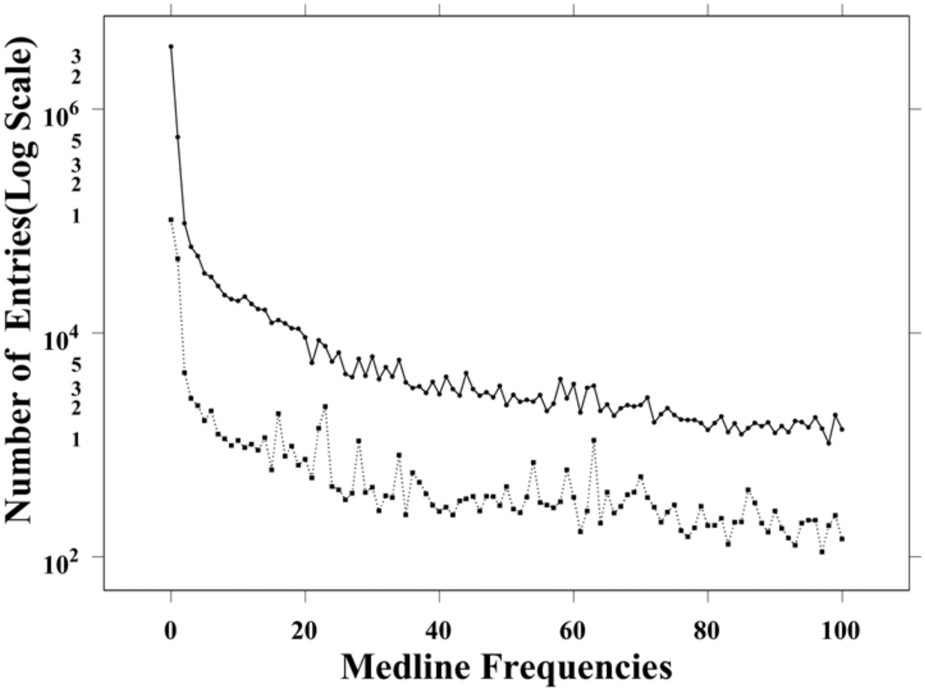

Figure 2.
Number of query terms at different frequencies in PubMed data that are followed by a lexically close term of ten times the frequency (solid curve) or preceded by a close term of ten times the frequency (broken curve).

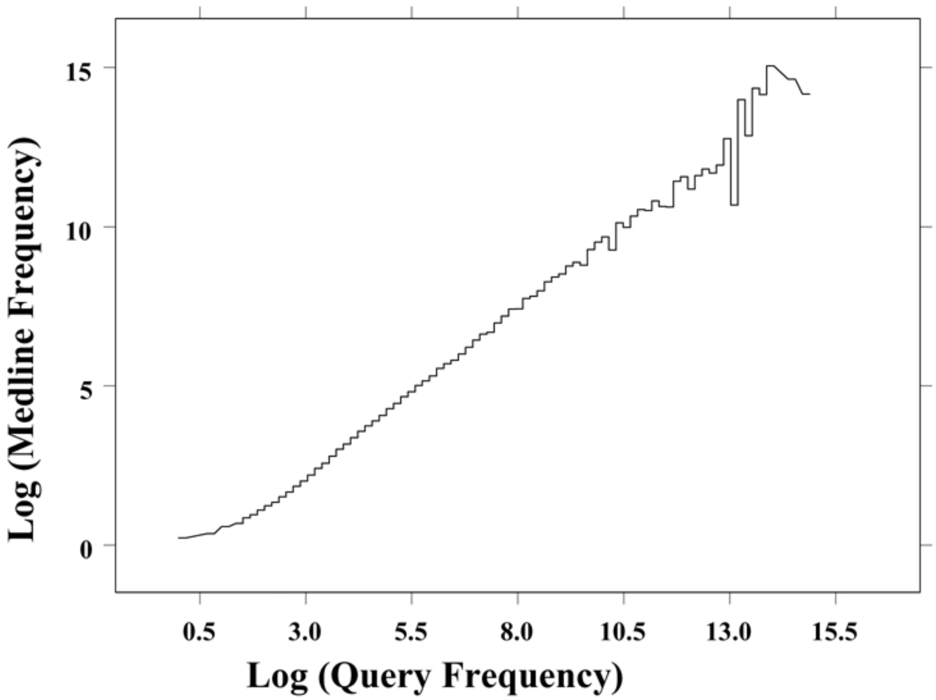

Figure 3.

Query terms are binned by log(query frequency) along the x-axis and the average of the log of the MEDLINE frequency over each bin is plotted on the y-axis.

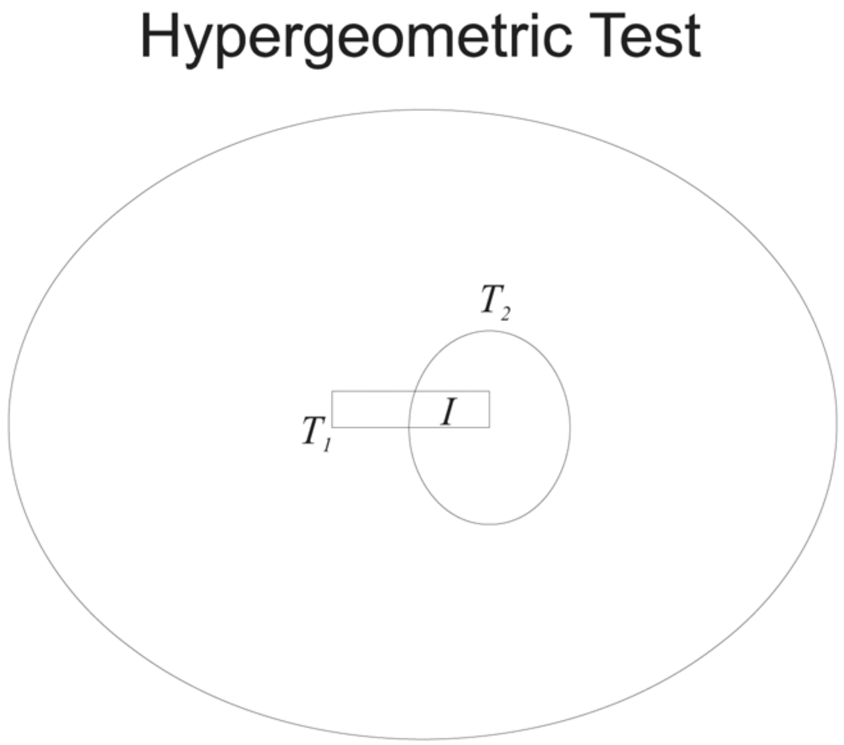

Figure 4.

In the space of all documents the rectangle represents the documents that contain the term T_1 and the small ellipse the set of documents that contain the term T_2. The intersection of these two sets is the overlap represented by I. The statistical significance of this overlap may be computed as the probability that this overlap is as large or larger than that actually seen assuming the two terms are nothing more than randomly related. This is known as a p-value and may be assessed by applying the hypergeometric distribution.

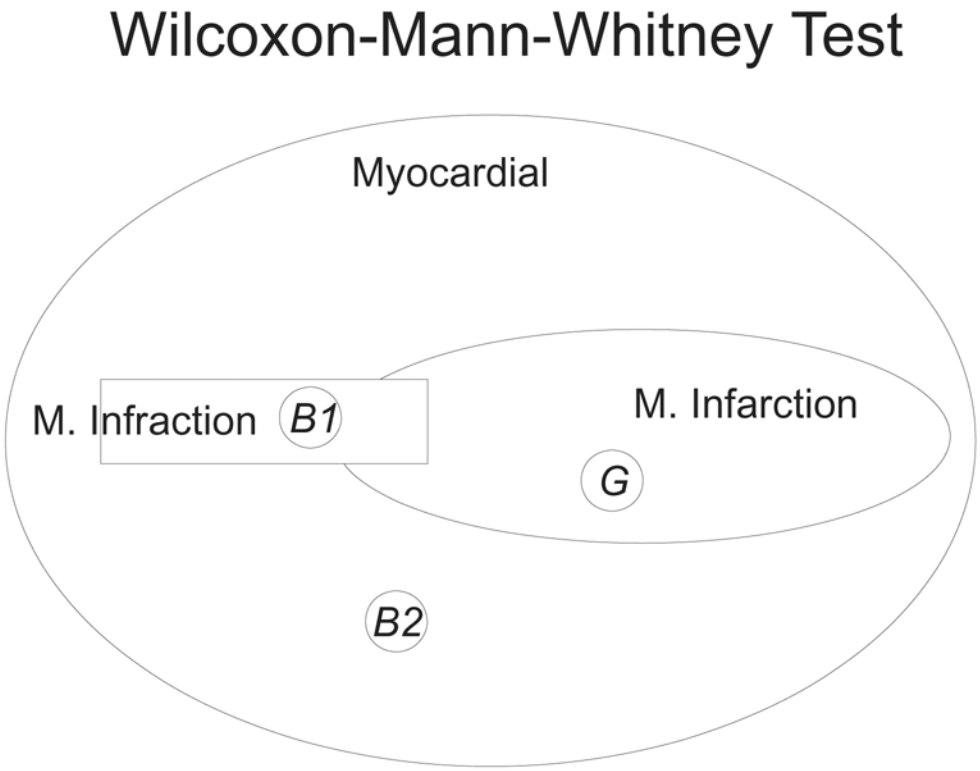

Figure 5.

We apply Naïve Bayesian learning to learn the difference between the positive set, labeled *G*, and the negative set consisting of the union of sets labeled *B1* and *B2*. From the weights learned we score both *B1* and *B2* and rank the union of the two sets. We then apply the WMW test to ask whether the sum of the ranks of the members of *B1* is higher than would be expected on a random basis. A *p*-value is computed to answer this question.

Table 1

Errors collected from 63 days of PubMed user logs.

	Number of erroneous words	Total number of edits
1 edit error	769128 (87%)	769128
2 edit error	105860 (12%)	211720
3 edit error	4932 (1%)	14796
Total	879920	995644

Table 2

For different word lengths the number of words sampled and edited to produce misspellings are shown as well as the percentage of such words the functions tried to correct and the percentage of success they had when a correction was suggested.

Single Token - single edit			
length	**Total Words**	**% attempts**	**% success**
3	1786	99	24
4	6701	73	45
5	10827	88	55
6	14823	86	72
7	13511	94	88
8	12330	98	90

Table 3

The same procedure as in the previous table except two edits are here introduced into each word.

Single Token - two edits			
length	**Total Words**	**% attempts**	**% success**
6	14408	78	13
7	13460	61	25
8	11938	96	65
9	10124	96	80
10	7974	95	87
11	5921	96	90

Table 4

The same procedure as in the previous table except three edits are here introduced into each word.

Single Token - three edits			
length	**Total Errors**	**% attempts**	**% success**
9	9485	84	39
10	7405	70	52
11	5496	69	67
12	4000	62	71
13	2835	59	76
14	1964	57	80

Table 5

On the left are some relatively common words and on the right the corrections suggested by the hypergeometric test. In many cases the words on the left are misspellings.

Non-optimal terms & frequencies		Corrections & frequencies	
acetycholine	153	acetylcholine	46852
acetycholinesterase	32	acetylcholinesterase	13207
acetyglucosamine	20	acetylglucosamine	4995
acetylate	287	acetylated	6594
acetylcholin	64	acetylcholine	46852
acetylcystein	64	acetylcysteine	3879
acetylocholine	20	acetylcholine	46852
acetylsalicyclic	157	acetylsalicylic	5186
achalasic	73	achalasia	2955
achatin	27	achatina	320
acheived	42	achieved	179735

Table 6

A sample of less than optimal query phrases on the left paired with their much higher frequency counterparts on the right. In some cases the phrase on the left involves a misspelling. In other cases it is simply not the most used form and hence would make a relatively poor query for the concept involved.

Non-optimal terms & frequencies		**Corrections & frequencies**	
myenteric neurone	9	myenteric neurons	593
myocardiac infarction	34	myocardial infarction	114638
myocardial infraction	122	myocardial infarction	114638
myocardial ischemic	870	myocardial ischemia	27214
myocardial necroses	77	myocardial necrosis	2055
myocardial revascularisation	234	myocardial revascularization	7343
myogenic expression	10	myogenin expression	119
myopia astigmatism	10	myopic astigmatism	276
myopia patients	19	myopic patients	231

Table 7

Examples of phrases that the spell checking algorithm processing PubMed queries can correct and the suggested corrections.

Misspelling	Correction given by algorithm
myocardial infraction	myocardial infarction
ear infraction	ear infection
miocardi alinfraction	myocardial infarction
terminl illnss	terminal illness
hig pressue liqud chromatogph	high pressure liquid chromatography
tumor necrosisactor	tumor necrosis factor
Hmgolbin	hemoglobin
Philariosis	filariosis

Table 8

Examples of mistakes made by the spell checking algorithm while processing PubMed queries.

Phrase	Mistaken correction
Sapna Baht	sauna bath
periostin	periods in
Daniel K E	danieluk m
bisexual molest	bisexual modest
pancreas & transplation	pancreas AND translation
stem cell ros	stem cell loss
Cupper hair	upper air