Streamlining text messages in a disaster management system: A Text Mining Approach

Maria Regina Estuar*
Ateneo de Manila University
Loyola Heights
Quezon City, Philippines
restuar@ateneo.edu

Hadrian Ang†
Ateneo de Manila University
Loyola Heights
Quezon City, Philippines
hadrianang@outlook.com

Miguel Palma‡
Ateneo de Manila University
Loyola Heights
Quezon City, Philippines
miguel.palma@obf.ateneo.edu

ABSTRACT

In developing countries, effectiveness of disaster management systems can be measured through adoption. To ensure that the general public embraces the technology, the design of the system should be technology inclusive. eBayanihan is a nationwide web - mobile participatory disaster management system which captures the human dimension of disaster by allowing ordinary citizens to post incidents as they experience it. This paper discusses possible solutions to problems encountered in the SMS based platform in eBayanihan. Specifically, we address the problem of correcting incorrect syntax. (we will place our solution here).

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

SMS, text mining, disaster systems

1. DISASTERS IN THE PHILIPPINES

More recently, an average of 19 typhoons enter the area of responsibility with around 6 to 9 making a landfall on Philippine soil [6]. In 2014, there was a total of X tropical rainstorms that hit the Philippines causing Y pesos of damage as well as Z loss of lives. In year, the Philippine government through Republic Act No. established the National Disaster Risk Reduction Management Council whose primary role is to enter role here. There are N clusters in this council ranging from enter clusters here. The cluster in Information Communications Technology (ICT) requires managing communication infrastructure as will as delivering information from top down (official news to the public) as well as bottom up (public information to the national level).

*Project Leader of eBayanihan system

†description

‡description.

2. REVIEW OF EXISTING DISASTER INFORMATION SYSTEMS

In the Philippines, there have been efforts in contributing to the development of disaster management systems and applications to collect and report disaster related information. Discuss systems here.

As of enter 2013, the Philippines received a worldwide rank of 12 in the mobile phone ownership with a ratio of enter ratios here. Since only 30 percent of the mobile users own a smart phone, there is still a need to provide an SMS based application for disaster reporting.

2.1 Types

2.2 Measuring Effectiveness

3. SMS BASED PLATFORMS

3.1 Design

The SMS based application was designed to accept reports based on the following syntax: enter syntax here.

enter technical description of algorithm for receiving, processing and plotting

enter figure of SMS feature phone and SMS app.

enter figure of SMS to eBayanihan

3.2 Problems

To send an SMS report to eBayanihan, a specific format has to be followed so that it can be parsed properly by the system. The SMS is then parsed for a keyword, urgency level, barangay, city or municipality and then the actual report. This information is then formatted as "POST <keyword>,<urgency>,<city> <barangay>,<remarks or message>" where text in brackets is replaced with the actual information for each field. Location information is then extracted from the SMS and used to map the report.

With this in mind, there are three problems that may occur with an SMS report. First, the SMS sent may not follow the proper format (there are extra or missing commas). Place example.

Second, keywords and urgency level may be misspelled, thus confusing the system. Place example.

Third, the location extracted may not be geo-locatable, which may occur because a) the names of places are misspelled, b) the place has not yet been mapped by the services used (Google Maps, Nominatim).

4. PROPOSED SOLUTION

There are two possible approaches in solving the problem, namely by correction and approximation. I will explain further here.

4.1 Correction

One possible approach to the problem of erroneous SMS reports is to attempt to correct them. A program first queries the database for new SMS that have been deemed to be wrong. Instead of separating solutions to the first and second problems mentioned above, the group instead decided to solve both simultaneously with the algorithm proposed in 4.1.1. Once the SMS report has been corrected for both spelling and formatting errors, it is once again passed through a regex check. If it passes, information is again extracted and sent via HTTP Post to the geolocation module of eBayanihan.

The group will first tackle the solution to the misspelled words problem and then explain how its approximate string matching solution can be adapted to solving the formatting problem at the same time. One approach to the misspelled words problem is to find the closest string in the given dictionary and then replacing the string with that one. Afterward, this may then be passed on to the gazetteer query stage. Numerous string similarity and distance metrics allow this kind of approximate matching. Hamming distance, Levenshtein distance, Damerau-Levenshtein distance, Longest Common Subsequence and Longest Common Substring are just some examples of these distance metrics.

Levenshtein distance, or edit distance, is a commonly used metric in approximate string matching and spelling correction. In Peter Norvig's trials, he claims that around 76% of errors were within an edit distance of one and 98.9% were within a distance of two [**?**]. It is defined as the minimum number of operations required to change one string into another. The operations defined are insertion (addition of an extra letter), deletion (removal of a character from the string), and substitution (replacing one character with another). Damerau-Levenshtein distance is an extension of this that keeps the three operations, but adds a fourth one, namely transposition or the flipping of adjacent characters. Damerau <cite Gonzalo Navarro Statistics>

Since there are only 15 keywords and 2 urgency levels, correcting errors given the dictionary is a simple task of selecting the word with the smallest edit distance when compared to a certain query. Since only a few comparisons are needed, a straightforward dynamic programming approach would actually work here.

$$d_{ij} = \begin{cases} d_{i-1,j-1} & for\ a_j = b_i \\ min \begin{cases} d_{i-1,j} + cost_{del}(b_i) \\ d_{i,j-1} + cost_{ins}(a_j) \\ d_{i-1,j-1} + cost_{sub}(a_j, b_i) \end{cases} & for\ a_j \neq b_i \end{cases}$$

To simplify computations, the cost of all operations will be equally set to one.

While the same approach may be used for the names of places, the dictionary here is a lot larger. With over a thousand cities, over a thousand municipalities and thousands of barangays, a straightforward comparison approach will be too slow (given the O(mn) run time of the dynamic programming approach). To solve this problem of efficiency, the group implemented a solution that involves a trie and dynamic programming.

Another problem arises though: what should the query string in approximate string matching be if the SMS is poorly formatted such that even data boundaries (delimiters, in this case commas, between fields such as keyword and urgency) are blurred? For this specific application, commas in the SMS format may be treated as delimiters to data fields, similar to white spaces in English words. This means correcting the format of the SMS may be reduced to finding boundaries between words. There is no absolute solution to the word boundary problem, but an approximate solution may be sufficient <CITE CHINESE GRAPH PEOPLE>. To solve this, the entire SMS is used as a query string and then a greedy approach that optimizes based on a scoring function decides when to extract tokens from the query. This is done iteratively until enough tokens have been extracted, as explained in the next subsection.

4.1.1 Trie with Dynamic Programming

A trie is a tree data structure where nodes can contain different keys pertaining to parts of a word. In this algorithm's case, each node contains a character. By using a trie, redundant computations are prevented, as prefixes to the wide range of words in the dictionary will only have to be involved in the computations once. In other words, the memoization array used in the dynamic programming algorithm is recycled, and carried over to other words with the same prefix.

<ADD PICTURE OF SAMPLE TRIE HERE>

Instead of simply using Levenshtein distance, the group instead opted to use the string optimal alignment distance function (sometimes known as restricted Damerau-Levenshtein distance). It also counts four operations instead of the three by the Levenshtein distance function, however, a single substring cannot be edited more than once. For example, the Damerau-Levenshtein distance of "there" and "etr" is two, as one can flip the first and second letters or "etre" and then insert an "h" in between. Its optimal string alignment distance , however, is three because after the initial transposition of the first two letters, one cannot insert another letter (the substring has already been edited). While more restrictive

than Damerau-Levenshtein distance, this metric is sufficient for the purposes of the algorithm (approximate matching of wrongly spelled words). This is what will be referred to from now on when edit distance is mentioned.

The first step to the algorithm is the construction of the trie. Since the the list of cities, municipalities and barangays is not expected to change very often, a separate program builds the trie, then outputs it to a text file to be read whenever the main algorithm has to be run. This will save time once the program is uploaded to the server, as the building of the trie is moved to the pre-processing stage.

When the main processing program is run, it first reads the trie and reconstructs it from a given text file. Once the full trie has been constructed, processing may begin. Let $s_1 = \{c_1, c_2, c_3...c_n\}$ be the entire SMS composed of n characters where c_i is the i_{th} character of the string. The string s_1 is then used as a parameter to a $computeDP(x)$ function where the parameter x is a designated query string. This function traverses through the entire trie in a DFS fashion, doing computations along the way.

Each node y contains two memoization arrays $Edit_y[n]$ and $Lcs_y[n]$ where n is the length of each array, equal to the length of the query string. These arrays hold the DP table for the corresponding operations according to the recurrence relations below. During this traversal, each node is processed. This processing involves setting its parent node, setting its depth, and then the computation of $Edit[0, 1...n]$ and $Lcs[0, 1...n]$ based on its parents. Each node normally contains a single character of some word in the given dictionary used to construct the trie. Suppose one is making for a certain node k. Let $name(k)$ refer to the character contained in node k while $depth(k)$ is its depth in the trie and $parent(k)$ is its parent according to the DFS traversal. If $name(k) = \$$ then backtracking from $parent(k), parent(parent(k))...root$, appending the names of nodes along the way, will result in a word that is in the dictionary (though in reverse). We let $word(k)$ be the resulting word if we were to backtrack from k to the root of the trie.

<INSERT RECURRENCE RELATIONS HERE>

Since any node y such that $name(y) = \$$ denotes a word, then $Edit_{parent(y)}$ and $Lcs_{parent(y)}$ will contain the edit distance and longest common subsequence of the query string s_1 along every point $\{c_1, c_2, c_3, ...c_n\}$ when compared to $word(y)$. A scoring function $f(x, z)$ is then defined to produce a "goodness score" for candidate $word(y)$. As the algorithm computes for the LCS and Edit Distance of the query string and $word(k)$, entering values into the memoization arrays as necessary, it keeps a set of running maximum and minimum values to find an index $e : f(Lcs_k[e], Edit_k[e])$ is maximized, resulting in the value $score_k$. A global maximum $score_{max}$, Lcs_{max}, and $Edit_{max}$ are also kept. The value of $score_k$, is then compared with a running global maximum $score_{max}$. If $score_k > score_{max} \rightarrow cand_{best} = word(k) \wedge score_{max} := score_k$.

For the purposes of this study, a simple scoring function $f(x, z) = \frac{x}{z}$ was used, the ratio between the Lcs and edit distance. If the edit distance is zero, the score is automatically made to be infinity. If $depth(y) > length(s_1) + T$ for any node y where T is some constant threshold edit distance provided, the trie traversal may stop. This is because $depth(y)$ denotes the length of $word(y)$ and if that goes above he length of the query string plus the threshold, the proceeding candidate words cannot be approximate matches for the query. The point p in the query string at which the maximum score $max(f(Lcs[0, 1, ...n], Edit[0, 1, ...n])$ was found is stored for the next step in the algorithm. We call p the cut point for this iteration.

Once the trie has been traversed (at least up to the threshold) and a global maximum score has been computed, the initial query string, s_1 in this case, is then cut at point p. The $word(y)$ that resulted in the maximum score is then appended to the answer string. The string $s_2 = \{c_{p+1}, c_{p+2}, ...c_n\}$ is then the new query string and is afterward passed into $computeDP(x)$ once again.This is done iteratively until the entire string has been processed (a cut is chosen at the end of the query string). Ties in score, edit distance and longest common subsequence are broken by taking the later index along the query string. For example, given the query string "therm" and $cand_{best}$ = "*there*" the cut point p will be after the last character "m." This is because both "ther" and "therm" have the same edit distance, longest common subsequence and score, when compared with the word "there." Because of this, we choose the cut point with the later index as mentioned earlier. This is to prevent the letter "m" from being matched with another candidate word, causing incorrect results (appending to words into the answer string, instead of only one).

The answer string is then returned for post-processing so it can be sent back to the geolocation program.

4.1.2 Critical Cases

Because the algorithm takes a greedy approach, maximizing based on only a simple scoring function, there are certain cases when it fails to correct the SMS string. One of the caveats is a result of the tie breaking measures taken, namely, choosing the later cut point. For example, we have the query string "earthquakMandau," a misspelling of the keyword "earthquake" concatenated with the incorrectly spelled "Mandaue," the name of a city in the Philippines. Both these mispellings "earthquak" and "Mandau" are only one operation away from their correctly spelled versions; however, they have been concatenated with no delimiter between them. When the algorithm runs, it successfully finds "earthquake" as the best candidate word, given its high LCS value and small edit distance. A problem occurs with regard to the cut point as both "earthquak" and "earthquakM" have the same score, LCS and edit distance when compared with the best candidate, "earthquake." Because of this, the cut point with the later index is chosen and we cut "earthquakM" out of the query string for the next iteration.

For the second iteration of the algorithm, "andau" will now be used as the query string. Because of the mistake made in the first iteration though, the best candidate is now "anda," the name of a municipality. This is because "andau" compared to "Mandaue" will result in an edit distance of 2 and an LCS of 5. Using the simple scoring function mentioned above, "Mandaue" will have a score of $\frac{5}{2}$. When compared

with "anda," however, the edit distance will be 1, the LCS will be 4, and with the same function, the score will be 4. Since $\frac{5}{2} < 4$, we select "anda" as the best candidate, an incorrect selection caused by a previous iteration.

Aside from this, the large number of cities, municipalities and barangays all in one trie can mean that even an edit distance of one can most likely sway the algorithm towards some other candidate. For example, typing "Manil" instead of "Manila," the capital of the Philippines, will result in an answer of "Manil" a barangay in the Southern Philippines. This is because the algorithm strictly checks for spelling and does not take into account other clues that may be in the message or the fact that Manila is a city and Manil is a barangay. The algorithm simply attempts to find the best possible match regardless of its context within the message (even if "Manil" is entered into the city field of the SMS).

4.2 Approximations

Text mining is proposed approximation approach in solving the problem. In this approach, the syntax is almost disregarded as input string is tokenized and compared to a lookup table for matching. The difference in this approach is that there is a validation process that happens in every correct or incorrect match so there is a percentage increase or decrease in the matching or relevance score.

We will use a machine learning classification algo here to tag correct and incorrect match.

5. EXPERIMENTATION

5.1 Problem1: Incorrect syntax

5.2 Problem2: Incorrect keyword or location

5.3 Problem3: Location not found

6. RESULTS

7. DISCUSSION

7.0.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[5]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

7.0.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

7.1 Citations

Citations to articles [1, 3, 2, 4], conference proceedings [3] or books [7, 5] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [5]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[5].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed.

7.2 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with

a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

7.3 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** and **.ps** files to be displayable with LaTeX. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption.

Note that either **.ps** or **.eps** formats are used; use the `\epsfig` or `\psfig` commands as appropriate for the different file types.

7.4 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

and don't forget to end the environment with figure*, not figure!

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)}\right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[7] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TeX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TeX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LaTeX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

8. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

9. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

10. ADDITIONAL AUTHORS

Additional authors: John Smith (The Thørväld Group, email: `jsmith@affiliation.org`) and Julius P. Kumquat (The Kumquat Consortium, email: `jpkumquat@consortium.net`).

11. REFERENCES

[1] M. Bowman, S. K. Debray, and L. L. Peterson. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.*, 15(5):795–825, November 1993.

[2] J. Braams. Babel, a multilingual style-option system for use with latex's standard document styles. *TUGboat*, 12(2):291–301, June 1991.

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

Figure 1: A sample black and white graphic (.eps format) that needs to span two columns of text.

[3] M. Clark. Post congress tristesse. In *TeX90 Conference Proceedings*, pages 84–89. TeX Users Group, March 1991.

[4] M. Herlihy. A methodology for implementing highly concurrent data objects. *ACM Trans. Program. Lang. Syst.*, 15(5):745–770, November 1993.

[5] L. Lamport. *LaTeX User's Guide and Document Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[6] N. of Author MI. Shoemaker.

[7] S. Salas and E. Hille. *Calculus: One and Several Variable*. John Wiley and Sons, New York, 1978.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chockfull of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.