Incorporating an Error Corpus into a Spellchecker for Maltese

Mike Rosner, Albert Gatt, Jan Joachimsen, Andrew Attard

University of Malta, University of Malta, University of Malta, University of Malta
mike.rosner@um.edu.mt, University of Malta, University of Malta, University of Malta

Abstract

This paper discusses the ongoing development of a new Maltese spell checker, highlighting the methodologies which would best suit such a language. We thus discuss several previous attempts, highlighting what we believe to be their weakest point: a lack of attention to context. Two developments are of particular interest, both of which concern the availability of language resources relevant to spellchecking: (i) the Maltese Language Resource Server (MLRS) which now includes a representative corpus of c. 100M words extracted from diverse documents including the Maltese Legislation, press releases and extracts from Maltese web-pages and (ii) an extensive and detailed corpus of spelling errors that was collected whilst part of the MLRS texts were being prepared. We describe the structure of these resources as well as the experimental approaches focused on context that we are now in a position to adopt. We describe the framework within which a variety of different approaches to spellchecking and evaluation will be carried out, and briefly discuss the first baseline system we have implemented. We conclude the paper with a roadmap for future improvements.

Keywords: Spellchecking, Maltese, Benchmarking

1. Introduction

Spellchecking is one of those problems which tends to be neglected under the assumption that it is easy to solve. As it turns out, even for highly resourced languages, this assumption is false. Accurate spellchecking is surprisingly hard, depending not only upon a comprehensive wordlist, but on as yet undetermined amounts of contextual information. For Maltese, it is particularly hard not only because resources are lacking, but also because of its mixed nature that incorporates Semitic and Romance substrates. One consequence is that word formation is complex incorporating both concatenative and non-concatenative morphological processes. In addition, there are many words in Maltese that admit spelling variants, partly as a result of the higher than average number of loan words that are found in Maltese.

2. Background

2.1. Maltese Word Formation

Being a Semitic language, Maltese has a lot of word forms which are derived and inflected non-concatenatively, i.e., word forms are changed internally according to the Semitic root-and-pattern structure: lexemes are derived from a mostly three-consonantal root, e.g. *k-t-b* for everything connected to writing. The derivation of specific lexemes like the verb "to write" or the noun "writer" of this root takes place by the application of certain patterns to the root consonants. The form *kiteb* "he wrote", for example, has the pattern **1v2v3**, in which the numbers stand for the root consonants and **v** for the vowels between them. The agent noun *kittieb* "writer" is formed by applying the agent noun pattern **1v22ie3**, in which the second root consonant is doubled and the vowel positions are filled with a short vowel and the long vowel *ie*, respectively. The following examples under 1 show several inflected verb forms of *kiteb* (examples (a)-(c)) and some nouns (examples (d)-(e)), all of which are based on the root *k-t-b*. The phonological transcriptions in IPA are not important at this stage; we will come back to them in section 2.2..

(1) (a) ***k*i*t*e*b*** "he wrote" /ˈkɪ.tɛp/
(b) ***k*i*tb*u** "they wrote" /ˈkɪd.bʊ/
(c) *ji****ktb****u* "they write" /ˈjɪg.dbʊ/
(d) ***k*i*tt*ie*b*** "writer" /kɪt.ˈtɪːp/
(e) ***kt*ie*b*** "book" /ˈktɪːp/; ***kotb****a* "books" /ˈkɔd.bɐ/

As a contact language between Siculo-Arabic, Romance and English, Maltese also contains a large number of loan words, most of which are incorporated into the Semitic non-concatenative morphology as stem-based forms. The examples under 2 show verb forms based on the stem *ddajvj-* (derived from English *dive*), which are not changed internally:

(2) (a) ***i*ddajv***ja* "he dived" /ɪd.ˈdɐɪ.vjɐ/
(b) ***i*ddajv***jaw* "they dived" /ɪd.dɐɪ.ˈvjɐʊ/
(c) *ji****ddajv****jaw* "they dive" /jɪd.dɐɪ.ˈvjɐʊ/

Thus, if one wants to incorporate morphological models into a spellchecker for Maltese, one has to take into account both the root-and-pattern-based and stem-based system. The next subsection will present some language specific spelling rules of Maltese which could be implemented in a rule-based spellchecker.

2.2. Some Specific Spelling Rules of Maltese

In general, spelling rules in an orthography are conventionalised spellings *options*, which are based on several linguistic and non-linguistic principles. Thus, while some rules are based on the phonetics, phonology and morphology of a language, other rules are rather based on "visual" principles, as e.g. the avoidance of trigeminates, i.e. consonant

clusters made up of three of the same consonant[1], or etymological spelling rules. Since these principles can be in conflict with each other (e.g. phonological vs. morphological spelling options), an orthography system has to find a way to give some rules priority over others in a consistent way so that they "together in a hierarchy create stability" (Il-Kunsill Nazzjonali tal-Ilsien Malti, 2008, 2).

In this section, we will concentrate on some spelling rules based on phonological, morphological and etymological principles.

Following the phonological principle, every consonant phoneme is represented by one grapheme in the Maltese orthography. Short and long vowels mostly "share" their graphemes (i.e., *a* stands for both [ɐ] and [ɐː]), but the vowel length can be determined by the reader from a general stress rule (Akkademja tal-Malti, 2001, 145-148).

However, as can be seen in example 1, the orthographical forms do not reflect purely the phonological forms. Comparing the orthographic representations with the phonological transcriptions, it becomes clear that the orthography does not represent the phenomenon called *regressive assimilation*. By this phonological filter the last obstruent in a sequence restricts the occurrence of a directly preceding obstruent in terms of the feature [+/- voice]: if the last obstruent in a sequence is voiceless, a directly preceding obstruent must be voiceless as well. If the last obstruent in a sequence is voiced, its directly preceding neighbour must be voiced, too (see examples (b), (c) and (e)). In word-final position, the pause after a final obstruent has the same effect as a voiceless obstruent, which is why in word-final coda position only voiceless obstruents can be found (see examples (a), (d), (e)).

Thus, in the orthographic forms, the phonological spelling rule is overridden by another principle, i.e., morphological constancy. It demands the spelling of word forms in such a way that they can be identified as members of one paradigm. For Semitic root-based forms as in 1, the root consonants (and their patterns) provide the common denominator by which the reader can recognize the underlying lexical form of the paradigm. For non-Semitic stem-based forms (see example 2), the underlying lexical form consists of a continuous stem.

A special case of morphological constancy in Maltese orthography is the etymological spelling of *my*, which represents a historic fricative that disappeared in most cases but left phonological traces on neighbouring vowels by lengthening them. Moreover, verbs containing *għ* as their second root consonant exhibit a stress pattern that differs from regular verbs and resembles verbs containing a sonorant (*l, r, m* or *n*) as their second root consonant (Fabri, 2009, 12). The examples under 3 illustrate these differences. While *niktbu* under (a) has the disyllabic pattern **'X.x**, the respective forms *nisirqu* under (b) and *nilagħbu* under (c) have the trisyllabic pattern **x.'X.x**. For *nisirqu*, this pattern is triggered by a phonological rule which prohibits complex syllable onsets consisting of *l/r/m/n* as first and a consonant as the second element and inserts a prothetic vowel *i* between them. For *nilagħbu*, the situation is a bit more complex. In the phonological form /nɪ.ˈlɐː.bʊ/, the trisyllabic stress pattern has survived while the consonant triggering it has disappeared. Following the etymological principle, this consonant is nevertheless written (as *għ*) in order to preserve the triconsonantal root *l-għ-b*. Also, following the principle of morphological constancy, this"virtual" consonant is written in the correct position where it formerly appeared. This allows the reader to correctly recognize the underlying root-and-pattern structure **ni1v23u**.

(3) (a) ***kiteb*** [ˈkɪ.tɛp] **1v2v3** "he wrote", *ni**ktb**u* /ˈnɪg.dbʊ/ **ni123u** "we write", *ni**kteb*** /ˈnɪk.tɛp/ **ni12v3** "I write"

(b) ***seraq*** /ˈsɛ.rɐʔ/ **1v2v3** "he stole", *ni**s**irqu* /nɪ.ˈsɪr.ʔʊ/ **ni1v23u** "we steal", *ni**sraq*** /ˈnɪs.rɐʔ/ **ni12v3** "I steal"

(c) ***lagħab*** /ˈlɐːp/ **1v2v3** "he played", *ni**l**a**għb**u* /nɪ.ˈlɐː.bʊ/ **ni1v23u** "we play", *ni**lgħab*** /ˈnɪ.lɐːp/ **ni12v3** "I play"

For the examples given so far, a rule-based spellchecker would first have to detect whether a certain word form is a root-based or stem-based form. This could be done by an (annotated) word list or lexicon. Secondly, it will have to parse the internal structure of a word (e.g. *niktbu* as **ni123u**) and compare it against a list of verbal paradigms with abstracted root-and-pattern structures. For verbs with "regular" consonants (as *kiteb*), if the respective structure **ni123u** is found in the list, the word form will be evaluated as correct. An incorrect form **nikitbu* will be recognised as incorrect, since the pattern ***ni1v23u** is not part of the relevant paradigm[2]. For verbs, however, that have *l, r, m, n* and *għ* in the place of **2** in the parsed pattern, a structure like **ni1v23u** would be possible. Therefore, the parsing process of the words would also need to include a rule that assigns these word forms to that specific verb class and evaluates the pattern structure of the candidate against the correct paradigm.

2.3. Maltese Spell Checking to date

There have been a number of previous attempts at developing a spellchecker for Maltese, but a good solution has so far proved elusive. Mangion (Mangion, 1999) implemented a set of rudimentary editing operations which were applied to a small, manually-created lexicon to see if they matched a candidate word. The main weakness was the small size of the lexicon, and to counter this, Mizzi (Mizzi, 2000) developed a lexicon-free system based on statistical n-gram language model which rejected words containing

[1]The preposition *minn* "from, by" is assimilated and reduced before the article *l-* to *mill-*, as in *kien intbagħat mill-Ministru* "he was sent by the Minister". Before a noun that starts with *l*, the combination of preposition and article is reduced to *mil-*, as in *joħroġ mil-labirint* "he went out from the labyrinth" (not **mill-labirint*).

[2]At this stage, we do not consider the recognition of possible similar patterns of other verb classes which could be offered as correct options instead of the incorrectly spelled form. Thus it might be possible to recognise **nikitbu* as the inversion of *in-* to *ni-* in the existing form *inkitbu* "they were written", where *in-* is not the 1st person prefix for "we" but a passive marker of the derivative passive paradigm.

n-grams whose probability was below threshold. This suffered a different problem: poor precision and recall. The best system to date is that implemented by R. Casha on behalf of the local Linux user group. This uses a large, partly rule-generated wordlist containing some 800K word types, employs the Unix `aspell` algorithm can be incorporated as a plug-in into various open-source tools such as Firefox and Open Office. It is deployed over the web[3].
This last system ticks many of the boxes but as a matter of fact is not very widely used, despite its availability and the clear need for a spellchecker. We believe that the main reason is that its performance simply isn't up to scratch, but this assertion remains a hunch, since its performance has never been systematically established.

3. Aims and Objectives

Given the shortcomings highlighted in the last section, we intend to develop a new spellchecker for Maltese with the following objectives in mind:

- **Wide coverage of the language.** The system has to be of clear benefit for the ordinary user and hence has to be general purpose and not oriented toward specialist document types.
- **Accuracy.** The system must offer a useful level of accuracy according to objectively formulated criteria.
- **Ease of integration.** We are not aiming for a laboratory prototype, but a system that is open source and easy to integrate into commercial software (such as Microsoft Word).

4. New Resources

Over the last couple of years, an important step towards developing language technology resources for Maltese has been the development of a corpus of Maltese texts, which is now publicly available from the Maltese Language Resource Server (MLRS)[4]. The corpus, which consists of written text, was collected opportunistically from publicly available documents and web pages, and currently contains ca. 100m tokens, falling into the following general topic areas (i) academic writing; (ii) laws; (iii) literature and criticism; (iv) transcribed parliamentary debates; (v) press; (vi) speeches (written to be spoken); (vii) miscellaneous texts from the web, including blogs and wikipedia articles.
Currently, corpus texts have structural markup (at paragraph, sentence and token level) and work is underway to complete a part-of-speech tagged version, to be released shortly. Meanwhile, the corpus itself functions as an invaluable resource for the development of spellchecking technology.

4.1. Error Corpus

During the process of opportunistic text collection for the MLRS corpus, one of the major problems encountered was the variability in the quality of the orthography of the various texts. First, some texts are written without using unicode-compliant Maltese characters (for example, a word like *ħaġa* 'thing/object' might be written as *haga*). This is a common problem with legacy data in Maltese produced prior to the widespread introduction of the unicode standard. Second, there is widespread variation among authors on the spelling of borrowed words. While some of these are well established and now have a 'Maltese' spelling (e.g. *futbol* 'football'), several others are either spelt using the original foreign (often, English) spelling, or adapted to Maltese orthography on an *ad hoc* basis. This is a situation which is expected to be rectified to some extent in the near future, given the recent publication of a set of guidelines for the spelling of foreign borrowings in Maltese. Finally, there are of course spelling errors of the more usual kind, due to typos etc.

Figure 1: Web interface for the spelling correction tool

As a result of these issues, we have undertaken a large-scale spelling correction exercise, which is now close to completion, and which involved manually inspecting every alphanumeric type in the corpus and offering a correction where one is perceived to be required. This exercise is being carried out by 60 students of the University of Malta, all of whom are completing a postgraduate diploma in proofreading. The exercise is being carried out as follows. The list of unique types in the corpus was divided into 30 roughly equal chunks of ($N \simeq 15000$ per chunk). Each chunk was allocated to two independent proofreaders. Proofreaders inspect types via an online interface. Through the interface, which is displayed in Figure 1, a proofreader inspects each type individually, with the option of inspecting the type in the various contexts in which it is found in the corpus. For each type, a proofreader has the option of (i) marking the type as correct; (ii) correcting the type in case of spelling errors (including cases where borrowings are spelt using Maltese orthography but do not conform to the guidelines); (iii) marking the type as a foreign word if it is simply used in its original spelling.
The outcome of this exercise is a database of approximately 450,000 types which have been inspected and corrected where necessary by two independent, trained proofreaders. Apart from the immediate benefit of being able to correct spelling errors in the MLRS corpus, the database is also a resource for (a) identifying common types of spelling error (e.g. common types of transposition errors and their common positions within the word); (b) supplying training data for statistical spellchecking algorithms.

[3] `http://linux.org.mt/projects/spellcheck/`
[4] See: `http://mlrs.research.um.edu.mt`

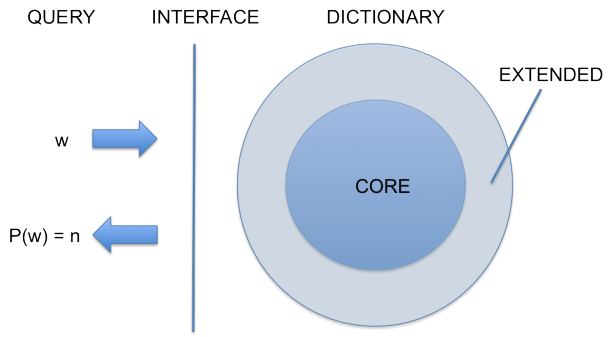

Figure 2: Dictionary

5. Current Approaches

In her classical review paper, Kukich (1992) views research in automatic spelling correction as focusing on three increasingly broad problems: (i) nonword error detection (ii) isolated word correction and (iii) context dependent word correction. Although all three have a role to play in an elaborate spelling correction system, it is clear that each can be studied and evaluated separately.

5.1. Nonword Error Detection

In general, error detection is a very hard problem to address because it involves the recognition of a mismatch between an abstract intention (what the writer intended) and what actually appeared. A subset of errors are nonwords whose detection does not involve intention, but merely decisions about wordhood. These are in principle easier to detect, requiring only a computable definition of wordhood. Most early approaches (e.g. The Unix `spell` program McIlroy (1982)) defined an error string as one not contained in a dictionary, so efficient coding of such a dictionary was seen as the core problem. The main weakness of this approach is that because the dictionary is never complete, the system cannot distinguish between nonwords and words that happen not to be in the dictionary.
But we note in passing that a large dictionary per se does not automatically improve performance[5], we subscribe to the principle that the larger the dictionary, the better, as pointed out by Damerau and Mays (1989).
In trying to overcome limitations imposed by dictionary size, we propose a dictionary structure as as shown in Figure 2. The core strings are those that are *known* to be real words whilst the "extended strings are those to which for whatever reason a probabilistic estimate of wordhood applies (e.g. is composed of frequently occurring bigrams). The idea is to accommodate words that are not necessarily defined extensionally. (Sampson, 1989), for example, found that hyphenated variants, negative forms etc. were a problem for dictionary-based detection. We believe that it would be possible to introduce generation rules to generate such "extended strings so that would have a high probability of wordhood.

[5]Peterson (1986) remarks that a large dictionary could contain words which are rarely encountered and which are similar to common misspellings (e.g. "wont), resulting in undetected errors

5.2. Isolated Word Correction

After detecting that a word is incorrect, the type of correction being offered by the spell checker is more or less dependent on the kind of application the spell checker is being used in. The spell checker could be designed to either correct the misspelled word automatically or it could be designed to interact with the user, offering a list of possible suggestions.
Kukich (1992) organises the different techniques used to tackle this problem into six different classes:

1. Minimum edit distance;
2. Similarity key;
3. Rule-based techniques
4. *n*-gram-based techniques
5. Probabilistic techniques
6. Neural nets

And in many of the cases, the process of correcting a word can be seen as (i) generating a list of candidate corrections, and (ii) ranking this list.
Given the available resources, we decided to focus initially on a combination of minimum edit distance and probabilistic techniques. One of the reasons for this is that this particular combination is well documented, having been used by Kernighan et al. (1990) a program called *correct* which attempts to correct a list of rejected words by the Unix *spell* program.
When evaluating a rejected word, *correct* first generates a list of *known* words that differ from the typo *t* by a single insertion, deletion, substitution or reversal. This candidate list is then ranked by calculating, for each candidate

$$P(c)P(t \mid c)$$

where:

- $P(c)$ - the prior; is the probability of the word appearing on its own
- $P(t \mid c)$ - the likelihood; computed from four confusion matrices
  - $del[x, y]$
  - $ins[x, y]$
  - $sub[x, y]$
  - $trans[x, y]$

This calculation, based on Bayes' Theorem, is used to rank the suggestions by looking at the word itself and its occurrences within a corpus. However, this does not guarantee that the highest ranked candidate is the correct one. In order to better rank candidates, one should also look at the context in which this word appears.

5.3. Context Sensitive Techniques

Real word errors are a significant problem for two reasons. The first is that they are relatively frequent. According to some earlier studies (Peterson (1986), Mitton (1987), Wing and Baddeley (1980)), anything from 16% to 40% of spelling errors yield real words, so they cannot simply be swept under the carpet. The second problem is that they will not be discovered by any of the isolated word methods for nonword detection just mentioned and consequently the only way to proceed is to make use of context in some way. But how do we define context, what kind of information is available under a particular definition of context, and what class of real word errors can be solved using that information?

There are many answers to these questions, and no one answer will suffice. The study carried out by (Atwell and Elliott, 1987) quoted in Kukich's review article is revealing. Carefully analysing a small sample of errors, they attempted to establish relative proportions of different kinds of spelling error across documents from different sources including those by children, non-native speakers, and proof-read published texts. They identified three categories of real word error (i) those in locally invalid syntactic contexts, (ii) those in globally invalid syntactic contexts (iii) syntactically valid but semantically anomalous. Although the distribution varied widely according to text genre it turned out that a significant proportion of errors were detectable as local syntactic violations, such as "The study was carried out *be* John Black". This kind of error could be addressed by looking for improbable N-grams of POS tags. Deeper level syntactic errors concerning e.g. subject verb agreement may require a full syntactic parse, and even though this could prove to be computationally heavy, it is not totally implausible.

To handle errors of type (i) a *sine qua non* is a POS tag language model, and hence, the availability of sufficient quantities of POS training data. A POS-tagger is under development, and is currently reaching a performance of around 90%. This is expected to improve in the medium term, so the outlook is promising for this line of attack. Errors of type (ii) will take a little longer to address, since work on deep level syntactic analysis has scarcely begun.

It is errors of type (iii) which are perhaps the most challenging. A typical example would be "they are leaving in about fifteen *minuets*". Any native speaker will immediately recognise and correct this error and would have probably have difficulty distinguishing between the detection and correction phases.

Detection of the error presumably arises because of the low frequency of collocations associated involving the words "leaving, fifteen" and "minuets". It is relatively easy to see how correction might proceed along the lines proposed by Kernighan, checking the relative frequencies of the collocations "leaving, fifteen" and all words within a fixed edit distance of "minuets". Presumably a peak would be found for "minutes".

We are poised to investigate the frequency of collocations using the functionality associated with the MLRS server. What remains unclear at this stage is whether the 100M word corpus is sufficiently large to detect an interesting proportion of type (iii) errors. However this is fertile ground for future empirical research.

Figure 3: Evaluation Metrics (Reynaert (2008)

5.4. Evaluation Criteria

In order to make progress we need to evaluate performance, yet to date evaluation of current and past efforts on spelling correction for Maltese has been entirely absent. A cornerstone of our approach is therefore to establish a methodology of evaluation and to begin by establishing some baselines.

We propose to adopt the evaluation framework proposed by Reynaert (2008) which uses the metrics illustrated in figure 3.

The large box represents the set of words in a text. The non-target portion represents the correct words, the target portion the incorrect forms. The smaller box represents the words that are actually selected for correction by a spellchecker. So those that intersect the target words are true positives (TP), whilst those intersecting non-target words are false positives (FP). The remaining unselected non-target words are true negatives (TN) whilst the unselected target words are false negatives (FN).

Reynaert's scheme includes five different levels of evaluation that distinguish between (i) core correction, (ii) error detection (iii) suggestion of correction candidates (iv) N-best ranking and (v) first-best ranking. The first level is the easiest to implement, being the only one in which it is sufficient to evaluate on lists of errors only (i.e. the detection task does not come into it.

In this paper we report on level (i) since this is our starting point. However, the other levels of evaluation will be progressively taken up in future.

6. Architectural Framework

Much effort has been put into the system's design. This takes the form of a quasi-abstract framework promoting extensibility and reusability. Our work revolves around one main objective – improving the system's performance. Given the peculiarities of the Maltese language and the fact that there are no benchmarked results, it is currently very difficult to judge which methodologies works best for Maltese. Hence, in conformity with component-based software engineering prinicples, the proposed framework allows a developer to plug-in and experiment with different components in order to perform evaluate and arrive at an optimum level of performance.

A simplified overview of this framework is illustrated in Figure 4.

The main building blocks of this architecture include:

- Interfaces - through which the system's main components (i.e. the spell checker and the system evaluator) interact
- Components - each of which is responsible for a specific task

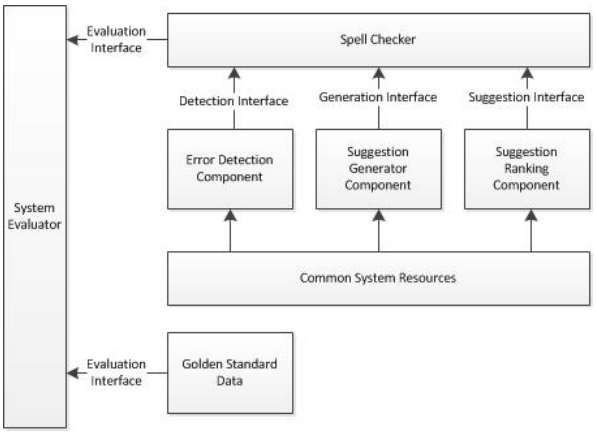

Figure 4: Overall Architecture

- A common repository of lexical resources - containing resources such as training data, dictionaries, and other resources which might be required by the system's components
- Gold standard data - used by the system evaluator componont to evaluate the spell checker's correctness

In order to better explain the above diagram, the following sections take into consideration the two main modules: (i) the spell checker, and (ii) the evaluator.

6.1. Spell Checker

A spell checker can be seen as an object which can make use of three components: (i) an error detection component (ii) a suggestion generator, and (iii) a suggestion ranking component. Each of these components could be invoked by a spell checker using the respective interface.
Each of these components is discussed in further detail in the following subsections.

6.1.1. Error Detection Component

Error detection is a fundamental module within a spell checker, whose functionality may vary according to the degree of sophistication of the spell checker itself. As described by (Kernighan et al., 1990), such a module may try to identify errors by considering a word in isolation, or taking into account the context in which it appears Even though the implementation of these may vary, they are both evaluating a string of characters. An instance of an error detection module will thus provide the system:

- A delimiter (such as a regular expression) of when the object should be invoked, and
- A degree of acceptance, ranging between zero and one

Rather than returning a boolean value, it would be better that the system returns a percentage of how *acceptable* this word is within our lexicon, allowing for more refined results and further processing. If the developer wishes to make use of boolean values, the result of this process could be either zero or one.

6.1.2. Error Correction

After a word has been tagged as an error, correction takes place; either by providing the user with a list of suggestions, or by automatically correcting the word. As to further modularize the process, the framework splits this into two separate processes, providing the respective interfaces:

- An interface for generating suggestions, and
- An interface for ranking the suggestions

Having these two processes loosely coupled allows for easier evaluation of different components as the evaluation system will be indicating the weak points of the system. Additionally the interface will require the components to implement

- A method *generate* which will be used to invoke the component responsible for generating the suggestions; taking as input a string of characters and returning a list of strings
- A method *rank* which will be used to invoke the component responsible for ranking the output list; taking and returning a list of string

6.2. System Evaluator

Spell checking can be either online or offline, and can be equipped with either automatic correction or it can be designed to interact with the user, or it may contain both functionalities. Consequently, the performance criteria can vary. For example a spell checker designed to interact with the user needs to be both robust as well as efficient, detecting the error and presenting the user with suggestions in a reasonable time. Hence, a developer might want to tune the spell checker to decrease execution time at the expense of correctness. The proposed framework anticipates different possible criteria, although to date we have only given serious consideration to the issue of correctness which is dependent on both error detection and error correction. The performance of error detection, as discussed in Renaert, can be calculated on true positives, true negatives, false positives and false negatives. Through the use of an *evaluation* interface, an evaluation summary is generated, given golden data. The output lists precision and recall, and the figures of the TPs, TNs, FPs and FNs.
Correction can then be evaluated on the generated suggestions, checking whether the correct word is in the list, and at what position of the list it is ranked.

7. Roadmap

The problem discussed in this paper remains open. However, we hope to have convinced the reader that the prerequisites are now in place for a series of development/evaluation cycles to be carried out, namely:

- a new collection of data resources, and the functionality to add more resources to the system;
- a framework which can be used to host different pluggable components which can be combined to work together to tune the spell checkers performance for the underlying application, and

MED	Count
0	268141
1	164665
2	68962
3	26626
4	10876
5	6071

Table 1: Error Distribution by Minimum Edit Distance

Cycle	Precision	Recall	F-Score
1	0.584	0.964	0.727
2	0.582	0.927	0.715
3	0.595	0.928	0.725
4	0.596	0.968	0.738
5	0.594	0.948	0.730
6	0.609	0.96	0.745
7	0.597	0.952	0.734
8	0.601	0.964	0.740
9	0.613	0.968	0.751
10	0.620	0.956	0.752

Table 2: Performance Figures

- consistent evaluation criteria.

In this section we put these elements together, forming a concrete roadmap describing what we have accomplished so far, and what we aim to achieve over time.

7.1. Establishing a Baseline

The first required task is to establish a series of baselines for each of the levels foreseen by Reynaert (2008). For this task we put together a very simple spell checker which takes as input a list of words (which can be either correct or misspelled) and outputs which words are recognized, along with a list of ranked suggestions for each unidentified word. Both error detection and correction focus on isolated words. Below we cover the resources and methods used to implement the spell checker, followed by the method used to evaluate its performance .

7.1.1. Error Detection

The error detection component makes use of an extensive wordlist kindly provided by R. Casha for the Linux User Group's online spell checker. This list contains proper names, nouns, verbs (including their inflections), and more. This is loaded in memory on startup, and is stored in a data structure using a hashing function. For each input word, the system computes its hashing function and if the entry is present in the data structure, then the word is considered to *exist* in the language. If however the word is not recognised, this is passed on to the error correction module.

7.1.2. Error Correction

Given a misspelled word, the error correction module generates a list of candidate corrections which is then ranked (as explained in section 6.1.2.). The list of candidate corrections is generated using the minimum edit distance (MED) function which is computed over all the words in the wordlist against the misspelled word. Those words which have a distance of one, are considered to be good candidates. This resulting list of suggestions is then ranked according to frequency in the MLRS corpus.

7.1.3. System Evaluation

At the time of writing we have only been able to cover level 1, for which we have prepared a series of test files from the error corpus. The distribution of errors by edit distance is shown in table 7.1.3..

The test file comprises a total of 500 words, randomly extracted from the hand corrected error corpus. The extracted words fall under two categories:

- a set of 250 correct words - having a minimum edit distance of zero
- a set of 250 misspelled words - having a minimum edit distance of one

This test file was given to the spell checker, and the output was checked with the hand corrected error data. Using this data, the TPs, FPs, TNs and FNs are calculated. This procedure was repeated ten times, randomly generating a different test file for each cycle. Precision and recall are recorded for every cycle, along with the corresponding F-Score. This set of results can be seen in table 7.1.3..

7.2. Future work

At the time of writing, we have only been able to implement a basic spell checker given available resources. However, as already mentioned new resources are becoming available as well as a framework which makes it easier for developers to create, share and tune the spell checker for the underlying application. The following subsections discuss the future plans.

7.2.1. Enriching the Spell Checker

Now that we have benchmarked a set of results, we plan to start improving the spell checker's performance by introducing new resources. From a linguistic perspective there is the possibility of introducing hand crafted well-formedness rules as suggested in section 2.2.. These will be used both during error detection (for evaluating possible inflections), as well as for error correction (to further enhance the ranking mechanism). Another exiting area for development concerns context-sensitive spelling correction. Enhancements to the functionality of MLRS using Sketch Engine (Kilgarriff et al., 2004) are ongoing. These are intended to yield data on the context of words in general that can be exploited in spell-chacking. In the longer term we expect to incorporate morphological analysis into the detection and correction processes, but research in this area is still embrionic.

7.2.2. Graphical User Interfaces

Having a framework in place might not be enough. The idea is to have two different graphical user interfaces (GUIs):

- a GUI for developers, which will provide drag and drop functionalities allowing the developer to easily test how different components behave together
- a GUI for users, which will provide a text editing environment

Additionally, inbuilt with the developer's GUI, we foresee the provision of functionality to export a current setup as a component which can be used in other applications.

8. References

Akkademja tal-Malti. 2001. *Regoli tal-Kitba tal-Malti*. Klabb Kotba Maltin, Malta: Valletta.

E. Atwell and S. Elliott. 1987. Dealing with ill-formed english text. In R. Garside, G. Leach, and G Sampson, editors, *Computational Analysis of English: A Corpus-Based Approach*, chapter 10. Longmans, New York.

F. Damerau and E. Mays. 1989. An examination of undetected typing errors. *Inf. Process. Manage.*, 25(6):659–664.

R. Fabri. 2009. Stem allomorphy in the Maltese verb. In T. Stolz, editor, *Ilsienna - Our Language*, volume 1, pages 1–20. Dr. N Brokmeyer, Bochum.

Il-Kunsill Nazzjonali tal-Ilsien Malti. 2008. Deċiżjonijiet 1 tal-kunsill nazzjonali tal-ilsien malti dwar il-varjanti ortografiċi. Il-Furjana Press.

Mark D. Kernighan, Kenneth Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. In *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 205–210. http://acl.ldc.upenn.edu/C/C90/C90-2036.pdf.

Adam Kilgarriff, Pavel Rychly, Pavel Smrz, and David Tugwell. 2004. The sketch engine. In *Proceedings of EURALEX*.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24:377–439, December. http://doi.acm.org/10.1145/146370.146380.

G. Mangion. 1999. Spelling Correction for Maltese. Technical report, Dept. CSAI, University of Malta, Msida MSD2080, Malta.

M. McIlroy. 1982. Development of a spelling list. *IEEE Transactions Communcations*, 30(1):91 – 99, jan.

R. Mitton. 1987. Spelling Checkers, Spelling Correctors, and the Misspellings of Poor Spellers. *Inf. Process. Manag.*, 23(5):495–505.

R. Mizzi. 2000. The Development of a Statistical Spell Checker for Maltese. Technical report, Dept. CSAI, University of Malta, Msida MSD2080, Malta.

J. Peterson. 1986. A Note on Undetected Typing Errors. *CACM*, 29(7):633–637.

M Reynaert. 2008. All, and only, the errors: more complete and consistent spelling and ocr-error correction evaluation. In N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odjik, S. Piperidis, and D. Tapias, editors, *Proc. LREC 2008*, Marrakech, Morocco. European Language Resources Association (ELRA). http://ilk.uvt.nl/downloads/pub/papers/477_paper.pdf.

G. 1989 Sampson. 1989. How fully does a machine-usable dictionary cover english text. *Lit. Ling Computing*, 4(1):29–35.

A. Wing and A. Baddeley. 1980. Spelling errors in handwriting: A corpus and distributional analysis. In U. Frith, editor, *Cognitive Processes in Spelling*. Academic Press, London.