Computational Linguistics in the Netherlands Journal 2 (2012) 2–22 Submitted 08/2012; Published 12/2012

Towards Identifying Normal Forms for Various Word Form Spellings on Twitter

Hans van Halteren, Nelleke Oostdijk
Dept. of Linguistics / CLST, Radboud University Nijmegen

Abstract

We take a first step towards the annotation of word forms in tweets with normal forms. Such annotation can assist research into spelling variation and the use of standard NLP tools to process tweets. This first step consists of the design of a technique to estimate whether two word forms can be considered variants of one and the same normal form. At this point we are examining word form types in isolation, i.e. without taking the context into account. We describe a word form similarity measurement which combines edit distance and context similarity over our whole tweet collection. Furthermore, we present the results of a pilot study, which we executed on 7Gw worth of Dutch tweets. We find that, while results are encouraging, various improvements to the similarity estimations are still possible.

1. Introduction

In recent years the use of social media such as chat, sms, facebook and twitter has taken enormous flight. While primarily these media are intended as means for communicating, sharing and exchanging information "there and then" with friends, family, peer group or members of some online community, they have also attracted the attention of various other parties for different reasons. Advertising agencies and businesses have been quick to realize that the social media constitute a rich data source which may be exploited, for example, for marketing purposes. From the available data one can learn which are the trending topics, what sentiments prevail, etc. Also, scholars from various disciplines (e.g. historians, sociologists, linguists) have discovered the wide possibilities that these data can offer, possibilities that before they could only dream of.

However, access to the data is hindered by the fact that, especially in these media, language users do not abide by the prescriptive rules as regards grammar and spelling that are generally upheld with standard language use in more conventional media (e.g. books, newspapers, magazines). Rather, the language used in the different social media appears to be developing into separate varieties, each characterized by numerous idiosyncratic grammatical structures and spelling variants. If this is indeed the case, we should not attempt to map it to the standard language by performing grammar and spelling correction. Still, for many purposes, such as text mining using standard IR/IE tools or research into spelling variation, it would be extremely useful if there were a view on the text in which the vast number of variants is reduced to more manageable proportions. Starting with spelling, and leaving the grammar for future research, we propose therefore to annotate each token in the text with some kind of *normal form*, which can act as a representative for a whole group of forms that are variants of each other, and for which we intend to adopt a consensual spelling (see further discussion below). For such annotation, it will be necessary, then, to measure whether different word forms might be variants of each other, so that they can be grouped together and a single normal form associated with them. We will attempt to do this measurement on the basis of an edit distance estimation and a context similarity estimation over a large data collection. In this paper, we test the basic usefulness of the two estimations, checking whether they lead to a likely normal form for a test set of word form types. Note that this makes the task different from spelling

correction, as we will be operating on the level of word form types rather than word form tokens in context.[1]

In the present paper we focus on the spelling encountered in Dutch Twitter data. The Netherlands is one of the top tweeting nations, with just over 22% of the Dutch online population using Twitter (November 2011; http://bigthink.com/strange-maps/539-vive-le-tweet-a-map-of-twitters-languages). Twitter is used typically for sharing information about what is happening almost instantaneously. Due to the medium, messages are limited to short texts (tweets) of maximally 140 characters. The vocabulary used in Dutch tweets is extremely rich: the tweets not only contain Dutch words and phrases, but also words and phrases from various other languages such as English or language varieties such as Dutch 'straattaal' ("street language"). Moreover, spelling variation occurs on a vast scale.

The structure of the paper is as follows. In the next section we first introduce the phenomenon of spelling variation in Twitter and then go on to define which normal form we will target in the context of this paper. Next, in Section 3, we describe the data which we will use to run and evaluate our experimental system. Section 4 describes how we estimate the edit distance and Section 5 how we estimate the contextual similarity and predict the most likely normal form. The results of the evaluation are detailed in Section 6. The paper concludes with a discussion and description of future work.

2. Spelling Variation in Twitter

Spelling variation found in tweets has several sources: as in any text, there are spelling errors but, other than in texts from conventional media, there is also quite a lot of spelling variation where the author deliberately chooses to deviate from standard spelling conventions (Tagg 2009).

When we consider the spelling errors we encounter in tweets, we find that these tend be of the same nature as those in traditional text. First, there are typographical errors, where the author in principle knows how a word should be spelled but makes a mistake in the operation of the text entry mechanism. However, such typographical errors display a wider range than is known from previous studies (e.g. Reynaert 2006) as these studies focused on errors made by users of traditional keyboards whereas tweets are often produced on smartphones with other keyboard layouts or even older types of mobile phones, possibly with algorithms (like T9) meant to assist text entry. Then, there are orthographic errors, where the author does not know the correct spelling (van Berkel and De Smedt 1988). As for these, they are known, also in more traditional text, to be strongly related to pronunciation. This relation is even more pronounced in tweets because of the more informal nature of tweets and the fact that Twitter language tends to be much closer to spoken language. For Dutch, this means we can expect effects like *n*-deletion (*inzette-inzetten* ('put in'); for *n*-deletion in Dutch spoken text, see e.g. (van de Velde and van Hout 2003) and confusion of phonemes which have various spellings (e.g. *ij-ei*: *batterei* instead of *batterij* ('battery')) or which are currently shifting in spoken Dutch (e.g. *z-s*: *jeself* instead of *jezelf* ('yourself'); (van de Velde et al. 1996)). Note that, for this paper, we will only pay attention to spelling. Errors like *gekoopt* instead of *gekocht* ('bought', with regular instead of irregular participle formation) or *enigste* instead of *enige* ('only', with incorrect use of the superlative) will be assumed to be errors against morphology and not against spelling, which makes them correctly spelled, albeit incorrect forms.

Spelling variation where the author deliberately chooses to deviate from what is the established correct form occurs quite often in tweets. There are various reasons for this. Because there is a limit on the number of characters in a tweet, authors are encouraged to use abbreviated spellings. This leads to shortened variants such as acronyms (*pww-proefwerkweek*, ('exam week')), devowelizations (*gwn-gewoon* ('just')), clipped forms (*eig-eigenlijk* ('in fact')) and rebus-like forms (*w8-wacht*

1. As far as we are aware, the task as we cast it here has not been investigated before. We do therefore not include a separate section on related work, but restrict ourselves to referring to previous work in the sections discussing the aspects of our work where such references are relevant.

('wait')). A good number of these shortened forms is already lexicalized, sometimes even inherited from IRC or SMS (*lol-laughing out loud; aight-all right*). Lengthened variants also occur, as words are lengthened to indicate stress. This lengthening occurs through the repetition of a single character (*jaaaaaaaaaaaaa-ja* ('yes')), a whole word (*nununununu-nu ('now'); lerenlerenleren-leren* ('learn')) or more rarely a shorter character sequence (*raarderder-raarder* ('stranger')). Most often the lengthening takes place in emotive words, but sometimes also in content words (*rustiggggg-rustig* ('quiet')). Finally, variant spellings may be used to indicate membership of a peer group. An obvious example here is dialect spelling, where the spelling represents the dialectal pronunciation (*woar-waar* ('where')). This type of spelling is known to be especially pronounced in Flanders (Vandekerckhove and Nobels 2010), but is certainly also present in the Netherlands, notably for the Southern and Eastern dialects. Here, we have to make a choice whether we want to recognize the dialectal spelling as a variant or as an independent form. Given our goal, viz. to assist research into spelling variation and to enable the processing with standard NLP tools, we have chosen to treat pronunciation-based variants of standard Dutch words as spelling variants, so that we will attempt to map them. However, if the word itself is dialectal rather than just the spelling, we do treat its variants as an independent group. An example is the dialectal noun *seule* ('bucket'), for which there is no pendant in standard Dutch. We recognize that there will be words where it will be hard to determine which type they belong to. For now, we will take a pragmatic approach. It is clear, however, that for the longer term there is a need for discussion to arrive at more principled guidelines.

The groups of spelling variants representing the same underlying form will normally be of modest size, some tens of potential variants, for frequent and/or long words extending into the hundreds, of which probably less than ten will be observed regularly, e.g. *politie* ('police') is seen regularly as *politie, politi, polit, polite, polisie* and *polietie*, but also as *politieee, poltitie, poilietie, poltiie, politieeee, pollitie, plitie, poitie, politsie, poliitie* and *politcie*. However, expressions of emotion giving rise to extensive repetition may have several thousands of forms in their cluster. For the underlying form *haha*, we have actually observed more than 2000 variants, including e.g. *haaaaaaaaaaaaaaaaaaaah-haaaaaaaaaaaaaahaaaaaaaaaaaaaaaaaaaahahaha*.

Conversely, observed forms tend to belong to only a few variant groups and usually just one for longer forms. However, here too we find exceptions, especially for shorter forms. An example is *ne* (Table 1) for which in a random sample of 100 tweets 11 different interpretations were encountered.

In further processing, a variant group which is represented by a list of all of its included variants is both unwieldy and difficult to interpret by humans. Therefore it would be preferable to select one variant form from within the cluster as the representative. Such a normal form should best be a form which can be taken to be a kind of consensus spelling. For most words, there are two basic choices for the consensus: the official spelling or the most frequent spelling. One might consider the official spelling a good target form. However, it has at least two disadvantages: it will sometimes be an exceptional spelling (e.g. *pannenkoek* ('pancake'), the official form today, is only half as frequent as the now archaic *pannekoek*) and sometimes does not even exist at all (e.g. for straattaal words such as *kaulo* ('asshole')). For the work we describe here, we will therefore organize our approach around the most frequent spelling, which also means that we have to center our work around the Twitter data and cannot simply start from existing lexica. In cases where different usage situations lead to different most frequent spellings, e.g. *vanavond* ('this evening') as *vanavond* in more formal tweets and as *vanaaf* in more colloquial tweets, we accept that we will see the creation of more than one variant group. Furthermore, in the current study, we restrict ourselves to the surface form and do not attempt to distinguish between different word classes and/or senses. Thus, although *loop* will probably show more spelling variation for the verb sense ('walk') than for the noun sense ('barrel of a gun') and *school* more for the sense 'a school for children' than for the sense 'a school of fish', there will still be only one variant group for each of the underlying forms *loop* and *school*.

Table 1: Interpretations for 100 random instances of the form *ne*.

Underlying form	Explanation of variant	Freq	Example
ne (abbreviation for school subject Nederlands)	Correct form, sense 1	16	Nu hw maken frans al gedaan bio ook nu ak maken xD en dan ak leren en ne leren
ne (foreign)	Correct form, sense 2	12	Het enige zinnetje wat ik in Frankrijk gebruikte was ' Je ne parle pa francais '
ne (dialect form of the indefinite article *een*)	Dialectal form	14	Die wisten gewoon da gij ne chinees waart
me	Typo: substitution	19	Ik ben ziek voel ne niet lwkkwe
en	Typo: transposition	9	dan maar leren voor anatomie ne engelss
nee	Typo: deletion (or deliberate shortening)	8	Oow ne he het regent
net	Typo: deletion (or deliberate shortening)	3	het leven is ne een rotblok soms komt komt het gewoon te hard
je	Typo: substitution	2	Fijn dat ne dat zegt als ik al weg beb
ene	Tokenization problem due to our tokenization process[2] (or deliberate phonetic spelling)	5	Geen 1ne moer zin meer om te werken
neem (cropped)	Tokenization problem due to Twitter medium	2	RT @NAME: Goede start vandaag , klant akkoord voor het uitvoeren van een ICT scan . Wil je meer weten over de QUBE ICT scan , ne ... [possibly: *neem contact op met*]
ne_t***	Tokenization problem due to typo by author	1	zit je ne top de computer moet je er weer van af
???	Unidentifiable	9	als k je niet ping ie ne ping

3. Experimental Data

In order to check the usefulness of the edit distance and overall context similarity estimations that we will use to determine whether two words are variants of each other, we need two data sets. First, we need a corpus, in this case of Twitter texts, to derive a context similarity estimation from. The corpus should be quite large, so that the estimation yields information that is exact enough to be used for the suggestion of the proper normal forms. Secondly, we need a list of word form types and their corresponding normal form, so that we can evaluate the normal form suggested by the system.

For a large corpus of Dutch language tweets, we were able to use about 608 million tweets from the year 2011. We are grateful to Erik Tjong Kim Sang who allowed us access to the data he collected (Tjong Kim Sang 2011). We tokenized these tweets, isolating the purely alphabetic tokens. All hash tags, user names, URLs, emoticons and such were mapped to representative special forms, e.g. @ for a user name. They were not subjected to the normal form suggestions, but were used when a token's context was investigated. Punctuation was separated from alphabetic tokens, as were digits. Fully alphabetic strings were left intact, also including clitic forms such as *kheb* ('I have'). This led to a total of about 7.4 billion tokens. For all processing in our system, we furthermore mapped all upper case letters to lower case and removed all diacritics.

For the lists of pairs of word form types and their corresponding normal form, we took a representative sample of all the types in the corpus described above. We restricted ourselves for this first experiment to fully alphabetic tokens (including apostrophes) and leave creative spellings such as *w8* (*wacht* = 'wait') for later.

Table 2: Type frequency bands in our corpus from which samples will be taken for system evaluation.

Frequency band	Examples	Total number of types	Coverage of alphabetic tokens in corpus	Cumulative coverage	Number of types in sample
> 1/1,000	*ik* ('I') *k* (*ik*; 'I')	150	60.2%	60.2%	150
> 1/10,000	*jammer* ('pity') *ak* ('geography')	724	19.6%	79.8%	724
> 1/100,000	*opgegeven* ('abandoned') *oowk* (*ook*; 'also')	3,947	10.8%	91.6%	1,024
> 1/1,000,000	*vuilnisbak* *watik* (*wat ik*; 'what I')	18,649	5.3%	96.3%	1,024
> 1/10,000,000	*aanstichter* ('instigator') *aaaardig* (*aardig*; 'nice')	80,542	2.3%	98.6%	1,024

3.1 Frequency dependent sampling

Since we wanted to investigate both frequent and less frequent types, we first divided the types into frequency bands, as shown in Table 2. The most frequent types occur in the tweets more than once per one thousand tokens. The next band holds types which occur more than once per ten thousand tokens, but do not already fall in the first band. We continue this band assignment up to types occurring more than once in ten million tokens. For the current experiment, types with lower frequencies were not used, as their overall context similarity estimate is likely to be too noisy for proper evaluation. Together, the five frequency bands we use cover only 26.6% of all observed types. However, these account for 98.6% of all observed tokens. We aimed for 1,024 sample forms per band. For the first two bands, there were fewer than 1,024 forms present and we included all of them in the sample. For the remaining three bands we made a random selection.

3.2 Manual annotation

We (the two authors) independently annotated all form types manually for the expected most frequent spelling variant (possibly the form itself). Where we disagreed, we came to a consensus by discussion. Annotation was largely on the basis of the type, but regularly a sample of instances in their contexts had to be inspected. During this exercise, it became clear that the language used on Twitter cannot be characterized as a single separate variety, but rather as a number of varieties. Given that it is hard to have an overview of all these varieties, but also their relative volume and therefore the relative frequency of the spelling variants in them, we have to assume that our annotation cannot be fully correct. However, we do think it is correct enough for use in an evaluation as presented here. Still, after applying our system, we decided to re-inspect 179 types where there was disagreement between our annotation and the system's suggestion, in order to check whether the system or we ourselves made a mistake. For 68 of these we indeed adjusted our annotation.[3]

In 32 cases we had interpreted the form correctly, but had misestimated what would be the most frequent form, usually selecting the orthographically correct form instead. This included a number of clitic forms, where the reduced form of the pronoun appears to be the most popular (e.g. *tis* ('it is'), *kwist* ('I knew'), *zegtie* ('he says'), *ofie* ('whether he')), although not always (*twel-hetwel* ('it' plus an adverb expressing confirmation)). Then there were plurals, diminutives and other words where the officially required apostrophe was most often left out (*msnen* ('msn'), *smsje* ('SMS'), *hollandsgottalent* (talent show)) but again not always (*bn'ers* ('Dutch celebrities'), *corona's*

3. These cases tend to be the more difficult ones and the percentage of errors here (38%) is far higher than the overall percentage of errors in our data set.

Table 3: Interpretations for 100 random instances of the form *omt*.

Normal form	Frequency	Example
komt	34	zo irritant op iemand wachten en ze omt nieeet
om het	27	Mn nagels zijn ook nog oranje gwn geen zin omt eraf te halen
om t_e***	12	Als j twitter gebruikt omt evertellen wat je doet is et saai
om	10	maar eigenlijk is het tijd omt naar huis te gaan
omdat	4	se heeft der hyves verwijderd omt iedereen der ging stalken
***k_omt	4	erk omt zo een tweet aanvan me moeder
om te	3	geen zin omt gaan
omt (name)	3	je doet mee aan het omt toch
ont_ploft	1	Wooow tank omt ploft nog geen politie
???	2	wat is omt boven op een berg ? – wtf ? Weet ik veel

Table 4: Classification of forms for differentiated evaluation.

Class	Number	Examples
Standard	3,013	ik, deze, gezegd, noodweer, ziennn
Jargon	70	rt, gwn, vanaaf, wss, ak (retweet, gewoon, vanavond, waarschijnlijk, aardrijkskunde)
Name	401	amsterdam, jan, whatsapp, lowlands, klm
Interjection	207	haha, xxx, omg, whahahaahahah, aaaaaaaaaaaaaah
Tokenization	100	kga, tvkijke, smiddags, kenningsmakings gesprek
Straattaal	45	facking, kaulo, wollah, gechapt, blaka
Foreign	110	cool, feature, vir, assasins

(the beer)). As could be expected, the normal confusion about forming past participles is visible also here, with some erroneous forms appearing to be more popular than their correct counterparts (*gespamt* ('spammed'), *gelult* ('talked nonsense')). A last recognizable subclass is formed by words where a schwa may be inserted at some point, which now also becomes visible in the spelling (*ongelofelijk* ('unbelievable'), *bijvoegelijk* ('adjectival')). The difference in frequency between the variant spellings can be very large, e.g. with *tis* 200 times more likely than *hetis* ('it is') and *daagjes* ten times more likely than *dagjes* ('days'). The variants can also be practically equally frequent, e.g. the already mentioned *ongelofelijk-ongelooflijk* and *bijvoegelijk-bijvoeglijk*, but also *noppes-nopes* ('nothing'), *pyjama-pyama* ('pyjama') and *snoeperd-snoepert* ('sweet tooth'). In such cases, we decided to choose the most frequent spelling for our experiment.[4] In the other 36 cases, we turned out to have misinterpreted the form. Usually this was a matter of ambiguity, where another sense than the expected one turned out to be more frequent in the corpus. For *late*, e.g., we expected that it would normally represent *laten* ('let'), as we saw rather many cases of *n*-deletion. However, a sample of 100 instances of *late* showed that 75 were in fact the form *late* ('late') itself, only 23 were *laten* and the remaining two represented *later* ('later') and an unidentifiable form (*late kei lang op ' D ' - sorry was ff eten*). More balanced cases were *douche*, with 59 instances of *douchen* (verb 'shower') and 41 of *douche* (noun 'shower'), and *ui*, with 44 times *uit* ('out') versus 40 times *ui* ('onion'). An example of very high ambiguity, only slightly less than *ne* shown above, is *omt* (Table 3). Apart from the ambiguous ones, there were also some forms where we had simply misunderstood the form. An example is the form *jaxxie*. We had originally interpreted this as a contracted form of *ja ik zie* ('yes, I see'), but it turned out to be a kind of diminutive variant of the name *Ajax* (the football club).

Apart from the expected most frequent spelling variant, we also annotated for the nature of each type (Table 4). The class standard contains all types not marked as one of the other six, special

4. We might just as well have decided to count both forms as correct but in this pilot we did not want to have special rules for ambiguous cases.

x / y	•	H	A	R	D	E	R
•	**0**	1	2	3	4	5	6
H	1	**0**	1	2	3	4	5
A	2	1	**0**	1	2	3	4
R	3	2	1	**0**	**1**	2	3
S	4	3	2	1	**1**	**2**	3
R	5	4	3	2	2	3	**2**

Figure 1: Example of the cost table constructed with the Levenshtein algorithm. We assume that the word *harder* is incorrectly spelled *harsr*, due to hitting the *s* instead of the *d* and leaving out the *e*. The algorithm correctly shows an edit distance of 2 operations, but also suggests an alternative sequence of operations viz. deletion of *d* and substituting *e* by s.

types; although called "standard", it also contains dialectal forms. We marked as jargon those forms which, although not necessarily part of the standard language vocabulary, appear to be accepted forms on Twitter. Note that this does not mean that they are not used outside of Twitter; some forms are inherited from the general language use of subgroups of authors, such as abbreviations for school subjects (*ak* for *aardrijkskunde* ('geography'), *gs* for *geschiedenis* ('history')). The jargon forms are almost always shortened forms. Under name we group all proper names and interjection is the class of tokens expressing some emotion but not participating in the syntactic structure of the tweet. The class with tokenization problems generally contains tokens which would normally be written as a series of tokens with spacing in between, e.g. *tvkijke* (*tv kijken*; 'watching TV') and *kga* (*ik ga*; 'I go'), but also a few half words that are part of a compound, which would normally be written as a single token in Dutch, e.g. *kennismakings* (mostly part of *kennismakingsgesprek* ('interview')). As the normal form for these forms, we chose the most frequent variant of the concatenated version. For example, *kga* will not be changed because this is the most frequent variant in clitic form, even though *ik* is more frequent than *k* when the two parts are written separately. The final two classes are straattaal (street language) and foreign words.

4. Edit Distance Estimation

Our starting point, and also the traditional approach, for the edit distance computation between two word forms is the Levenshtein algorithm (Levenshtein 1965; English translation 1966), using costs which vary per inserted/deleted character or substitution character pair. However, for the spelling variation we observe in Twitter, the Levenshtein algorithm does not suffice. The repetition of character sequences, as described above, forces us to use an algorithm which offers the possibility of moving backward in one of the two compared strings. And if we move away from the pure Levenshtein anyway, we would also like to be able to handle other cases that are problematic for Levenshtein, such as transposition (e.g. *omzet-ozmet* ('turnover')) and predictable substitutions involving longer character sequences (e.g. *nacht-nagt* ('night')). Our solution is patterned on the Viterbi algorithm (Viterbi 1967), another dynamic programming algorithm which is mostly known as the decoding algorithm for Hidden Markov Models. In the following sections, we will first describe the two algorithms we borrowed from and then our own algorithm, which we have dubbed Vitershtein.

4.1 Levenshtein and Viterbi

Most often, edit distance is calculated with the Levenshtein algorithm (Levenshtein 1965; English translation 1966). In this algorithm, two strings to be compared are (virtually) placed alongside a

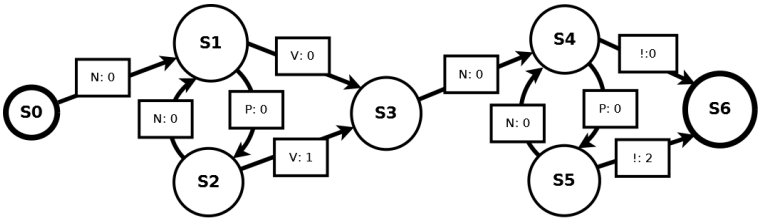

Figure 2: State network for simple newspaper headlines.

two-dimensional grid (see Figure 1). The grid will then be filled in such a way that the number in a cell [x,y] describes the edit distance between the substrings up to positions x and y. After all cells have been filled, the distance for the complete strings can be found in the bottom right cell. When filling a cell, the continuation from cells [x-1,y-1], [x-1,y] and [x,y-1] is considered. The first represents substitution. In the base version of the algorithm, the cost of this move is 0 if the characters on positions x and y in the respective strings are the same, and 1 otherwise. The latter two represent insertion/deletion and (in the base version) always cost 1. The cost of each move is added to the contents of the cell of origin and the lowest of the three values is placed in the target cell. The base version is usually adjusted in that the cost of various moves is based on their occurrence in observed corresponding string pairs, e.g. in speech-related comparisons, a schwa would have a relative low deletion/insertion cost and voiced and unvoiced versions of a phone would have a relatively low mutual substitution cost.

The Viterbi algorithm (Viterbi 1967) is also meant to determine a path of lowest cost (or alternatively highest probability). Where Levenshtein specifically targets the alignment of two linear strings, Viterbi is more general in that it tries to align any series of observations with a path in a network of connected nodes, which is usually described as a collection of *states* with *transitions* between them and with a *start state* and one or more *end states*. In Figure 2, we show an example of a network that models a number of (sensationalist) newspaper headlines like *Man With Spear Chased Bear In Wood!*. The transitions in the network are shown as arrows, each with a list of costs for the various parts of speech that can be observed on that transition. The headline has to start off with a noun, which can be followed by any number of preposition-noun pairs. Then we proceed to the verb, after which we again allow the same kind of noun phrase as before the verb. The headline is terminated with an exclamation mark. We also allow the preposition phrase to miss its prepositional complement, i.e. for *Man Without Found Woman With!*, but this is given a higher cost.

The fundamental observation underlying the Viterbi decoding algorithm is that a subpath of the best path for the complete series of observations is necessarily a best subpath on the stretch of the observation series that it covers. This means that for any subpath, one needs only remember the lowest cost to get there and the state from which a transition leads to that lowest cost (the *back pointer*). Therefore, all necessary information can be stored if one creates a list for every position in the observation series, in which for every state the lowest cost and the back pointer are noted. After the algorithm has collected all the information, the best overall path can be found by following back the chain of back pointers, starting at the end state.

These lists can be visualized as a so-called *trellis*. We show an example trellis in Figure 3, corresponding to the example headlines *Man On Plane Above Saw UFO!*, but showing only the paths that lead to non-infinite costs.[5]

When calculating the costs for all the state-position pairs in the trellis, Viterbi starts at the second line in the trellis (the first one having only the start state at cost 0) and works its way down

5. This is a very simplified example. In most applications of Viterbi, there are many more potential transitions and there is much more variety in cost.

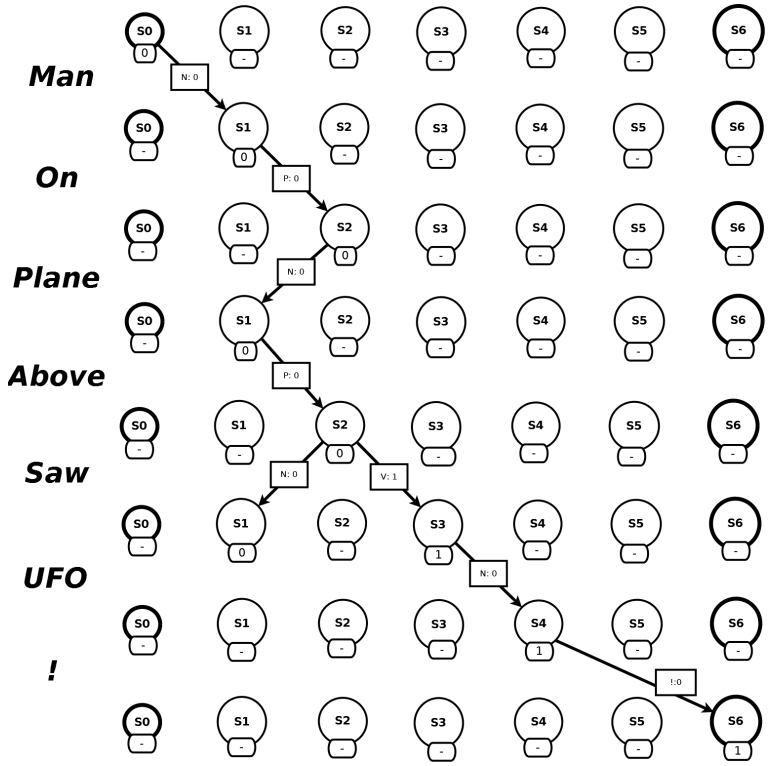

Figure 3: Example trellis for a headline being decoded with the network of Figure 2.

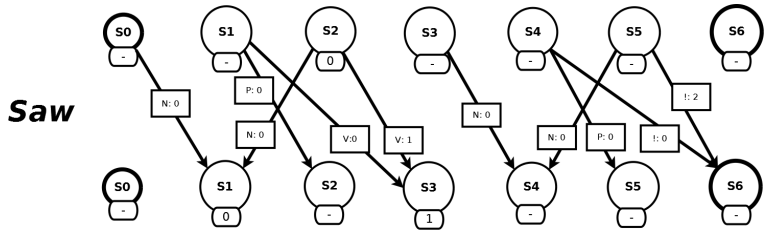

Figure 4: An example of one progression of Viterbi from one trellis line to the next.

line by line. For every line, it visits every state and checks the transition from every state in the previous line. If the cost at the origin and the cost of the transition are both less than infinite, the two costs are added and suggested for the target state. After checking all possible transitions to a target state, the lowest suggested value is stored, along with the corresponding back pointer. An example can be seen in Figure 4, where all possible transitions of the network are shown, linked to the observation of the word *Saw* in the decoding of Figure 3.

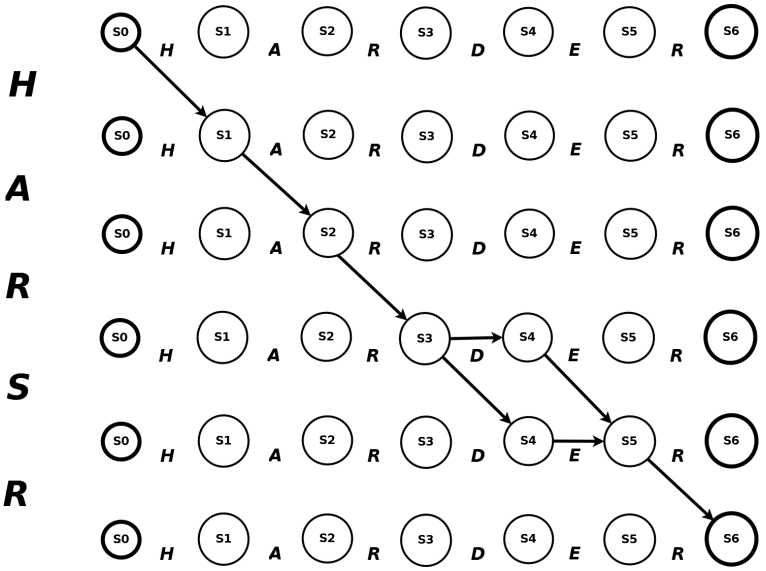

Figure 5: Example route along position-state pairs in Vitershtein corresponding to the Levensthein cost table in Figure 1.

4.2 Combining Levenshtein and Viterbi to Vitershtein

It is possible to recast the comparison done by the Levenshtein algorithm in terms of the Viterbi algorithm. We transform one string (the *model string*) into a network of states.[6] We create one more state than the length of the model string. The first (numbered 0) will be the start state; the last the end state. Each transition corresponds to one character in the string to be decoded. If the model string itself is decoded, this is meant to lead to a route through all the states in numerical order, which means that we can label the position between two adjacent states with the model string character that is matched on that transition on the model string route. The second string remains as it is, i.e. simply a series of observations, in this case characters, which will be linked to the transitions when the network is traversed. In Figure 5, we show the Viterbi trellis that corresponds to the Levenshtein cost table of Figure 1. When we introduce the various edit operations below, we will every time illustrate them by showing lines from this trellis and the transitions originating at S2, in the same format as used in Figure 4, as we think this will be clearer than showing the network without reference to the progression through the string to be decoded.

If we organize our network as described, what does this mean for the three possible moves in Levenshtein? The move from [x-1,y-1] to [x,y] is the most straightforward. It advances the position in the observed series one place and also takes one state transition. We merely need to set the transition cost for that transition to the substitution cost between the character observed and the character connected to that transition. If the two characters are the same, this should be 0. For other character pairs, for ease of explanation, we will be using cost of 1, but we will come back to this below. This is shown in Figure 6, where the substitution transition from S2 is shown (the arrow) together with the costs associated with every decoded character (the list in the box on the arrow).

6. In fact, we select the more frequent variant to be the model string. We will come back to this when we discuss the repetition operation in Section 4.3.

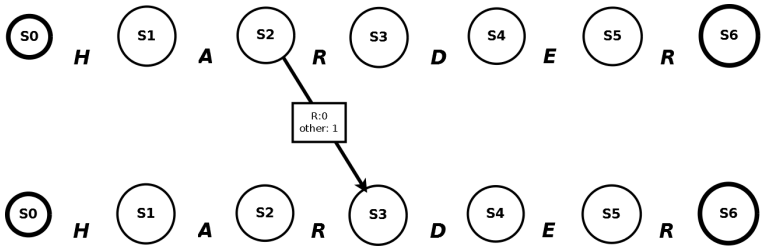

Figure 6: Example of the substitution operation in Vitershtein.

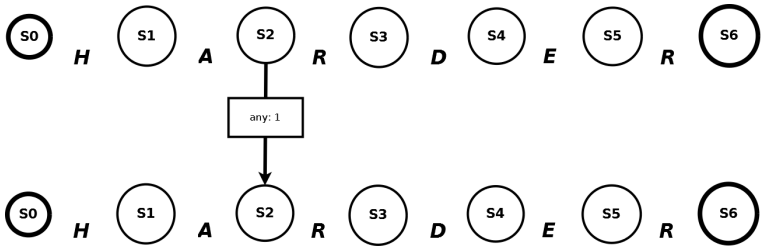

Figure 7: Example of the insertion operation in Vitershtein.

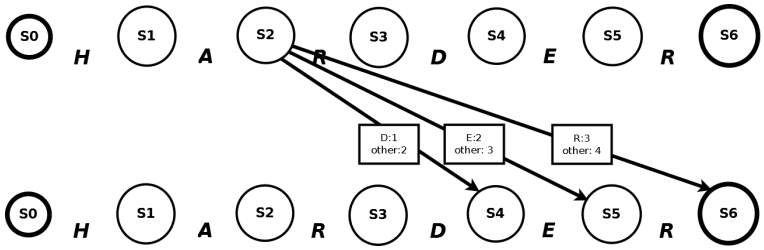

Figure 8: Example of the deletion operation in Vitershtein.

The move from [x,y-1] to [x,y] is an insertion into the model string. It advances the observation position, but stays in the same state (see Figure 7). Its cost is the insertion cost of the observed character, for this description again set at 1.

The move from [x-1,y] to [x,y], a deletion in relation to the model string, is the most complex. The problem is that the observation position is not advanced. However, all transitions in the network are connected to an advance of this position, since Viterbi proceeds line by line (cf. Section 4.1). This means we have to introduce transitions from each state to all following states, with a cost that is the sum of the deletion costs of all the skipped characters,[7] plus the substitution cost of the final character. Examples can be seen in Figure 8, which shows all possible deletion transitions from S2.

Where, in the explanation, all costs were 0 or 1, we differentiate the costs in our actual implementation in such a way that more likely variation is given lower cost. In principle, the cost for substitutions, insertions and deletions should be based on observed frequencies in variant pairs. However, as we lack the necessary data for this, we have for now only applied some adjustments on the basis of our intuition. We decided to override the standard cost of 1 only where we have a strong expectation of variation, namely

7. For this explanation again kept at 1 each so that the sum is the number of deleted characters.

- Substitutions of *v-f, s-z, d-t* (all 0.01) and *b-p* (0.1)
- Substitutions with a character adjacent on the computer keyboard (0.7).[8]
- Insertion/deletion of *h* (0.01), *e* (0.2) and *n* (0.2)
- Other deletions are cheaper when they reduce a series of equal characters (*undoubling*; 0.7)
- Insertions are cheaper if the inserted character is adjacent to the previous/next model string character on the computer keyboard (0.7).

With the three operations now described, and the example costs, the algorithm will decode *HARSR* as one of

- S0 [0] - *H* - S1 [0] - *A* - S2 [0] - *R* - S3 [0] - *S* - S5 [2] - *R* - S6 [2]
- S0 [0] - *H* - S1 [0] - *A* - S2 [0] - *R* - S3 [0] - *S* - S4 [1] - *R* - S6 [2]

just as Figures 1 and 5. However, with the intuitive costs, this becomes

- S0 [0] - *H* - S1 [0] - *A* - S2 [0] - *R* - S3 [0] - *S* - S5 [1.7][9] - *R* - S6 [1.7]
- S0 [0] - *H* - S1 [0] - *A* - S2 [0] - *R* - S3 [0] - *S* - S4 [0.7][10] - *R* - S6 [0.9][11]

and the system gives a lower cost to our originally intended typo scenario.

4.3 The added potential of Vitershtein

With Vitershtein, we want to overcome two limitations of Levenshtein. First of all, Levenshtein cannot move backward in either string, so that repetition cannot be handled. Secondly, Levenshtein can only apply edit operations that involve at most one character in each string, so that transposition cannot be handled. Staying within the scope of the original Viterbi, Vitershtein can handle repetition by allowing jumps to lower numbered states (i.e. states that correspond to characters earlier in the model string) at a cost which only represents the substitution of the last character.[12] Figure 9 shows all possible repetition transitions originating from S2.[13]

At this point we can see an asymmetry in the handling of the two strings to be compared. Where the repetition is made cheap in the observed string, repetition in the model string which does not occur in the observed string is taken to be normal deletion and is handled by jumping forward along the states, which has rather higher cost (Figure 8). Given the intended application, this is not a problem, as in our application we will always try to determine whether a more frequent and a less frequent string are spelling variants of each other and we can expect the more frequent string to be less convoluted than the less frequent one. However, when more symmetry is needed in the edit operations, we expect that a more generalized algorithm than Vitershtein will be needed.

For edit operations involving longer substrings than single characters, we even need to go beyond Viterbi and allow transitions that refer back further than the state cost list of the previous position in the observed series. Take as an example transposition. We can handle this in Vitershtein by way of a multi-position transition in the network (see Figure 10, which shows the transposition transition

8. As already mentioned earlier, various other input methods are also, if not even more often, used to produce tweets, which means that we should also investigate which other typing mishaps tend to occur more often.
9. Cost of 1 for insertion of the *D* and 0.7 for the substitution of the *E* by the adjacent *S*.
10. Cost of 0.7 for the substitution of the *D* by the adjacent *S*.
11. Cost of 0.2 for the insertion of *E*.
12. The additional cost is again kept simple for the purpose of easy explanation. Although it is here shown as 0, our implementation actually uses a cost of 0.001 to favour lower numbers of repetitions over higher numbers.
13. The transition to S2 does not have an "other" option as this has already been handled above as insertion (Figure 7). This does mean that the "any" there should rather be read as "other than A".

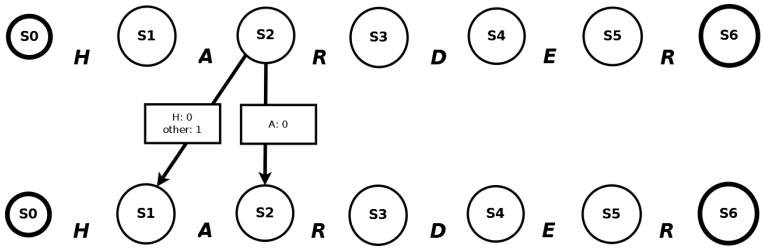

Figure 9: Example of the repetition operation in Vitershtein.

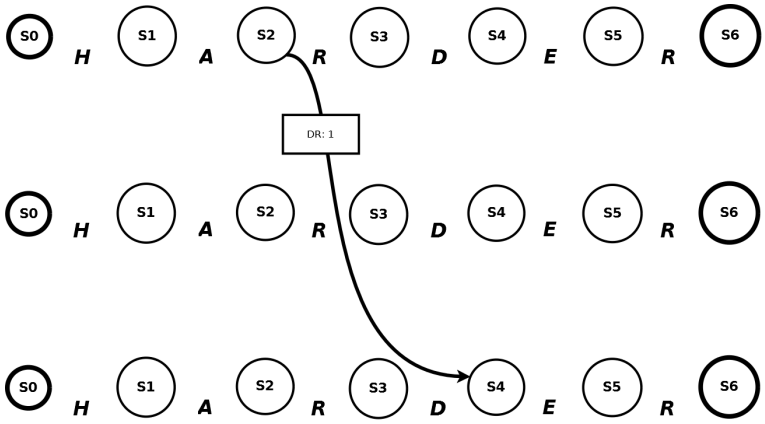

Figure 10: Example of the transposition operation in Vitershtein.

from S2). Now, the transposition by itself costs 1. However, the route via two substitutions is also still accessible and will be preferred if it has lower cost.[14] Multi-position transitions can also be used for mapping between other related substrings, e.g. string pairs like *ch-g* and *ij-y*.

On the algorithm side, the changes are not complicated. Where, in Viterbi, a loop was needed along all the states on the previous position, Vitershtein needs to loop over all incoming network links. In the current application, the number of multi-position transitions tends to be low, so that the speed of the algorithm is not noticeably reduced.

With the complete Vitershtein, even variants like *HARDERDREDER* will now yield a relatively low edit distance:

- S0 [0] - *H* - S1 [0] - *A* - S2 [0] - *R* - S3 [0] - *D* - S4 [0] - *E* - S5 [0] - *R* - S6 [0] - *D* - S4 [0.001][15] - *RE* - S6 [1.001][16] - *D* - S4 [1.002] - *E* - S5 [1.002] - *R* - S6 [1.002]

Our final tweet-inspired editing operation is not merged into the Vitershtein algorithm. We have observed that sometimes words are abbreviated by leaving out all vowels. Therefore, if one of the strings contains no vowels, we calculate two edit distances, namely a normal one and one in which all vowels are removed from the second string as well, and we use the lowest distance.

14. This is of course only possible once we are using variable cost instead of the default 1 point.
15. Cost of 0.001 for the jump back.
16. Cost of 1 for the transposition, which is lower both than 1 for the insertion of *R* plus 1 for the deletion of *R* and 0.7 for substitution of *E* by *R* plus 0.7 for substitution of *R* by *E*.

5. Automatic Selection of the Most Likely Normal Form

For each word form type, we want our system to find the most frequent form in its variant group. Such a form is expected to be similarly spelled, as it should be in the same variant group, and to be more frequent in the contexts that the original type occurs in. We can therefore go about finding the target variant by having the system examine each occurrence of the original type in the corpus.[17] For each occurrence, the system can then determine which other similarly spelled word forms are used more frequently in the same context.[18] Next the system has to choose which form is the best alternative overall by considering the suggestions in all occurrences. Especially when the type under examination is a more extreme variant of the normal form, this best alternative may itself also have received a better alternative, in which case this choice is chained. Finally, the system has to check if there are enough occurrences where an alternative is suggested. If not, the original word form is suggested as its own normal form.

5.1 Locally better alternatives

So, first, for each occurrence of a word form type, we want to determine whether there is an alternative "better" spelling variant in the current context. This means finding tokens which a) can be deemed spelling variants of the current token and b) are used more frequently in the current context than the token now present in that context.

For a), we compare the edit distance estimated with the Vitershtein algorithm (Section 4) with a threshold, currently set at one edit cost point for every five characters beyond the first in the suggested alternative.

For b), we first need to decide what we want to use as context. For the work described here, we used a simple window of two tokens left and right of the focus token, which is already quite large in relation to what is usual in other tasks, such as spelling correction, although extension of the window is already being looked into (e.g. Stehouwer and Van Zaanen 2010), and which generally also suffices in tagging. However, considering the size of the corpus we are using and our (currently) available computational resources, we could not afford to use the actual 5-grams. Instead, for each token, we used all three trigrams it participates in[19] and compare the frequencies of these trigrams with those of the trigrams we get when replacing the focus token with the suggested alternative. E.g. in the tweet *was echt leuuk om weer te zingen in den haag* ('was really nice to sing again in The Hague'), the token *leuuk* participates in the trigrams *was echt leuuk*, *echt leuuk om* and *leuuk om weer*. These trigrams are observed 828, 35 and 3 times respectively. For the desired alternative, *leuk* ('nice'), the trigrams *was echt leuk*, *echt leuk om* and *leuk om weer* occur 77871, 22956 and 8387 times in our corpus respectively. In this case, all three frequencies are higher and we can definitely say that *leuk* is a better variant. Since even trigrams will be relatively sparse, we do not demand all three frequencies to be higher, but only two of the three. However, these two frequencies both have to be higher by a factor of at least 2. Note that, in this example, this leniency of not demanding three higher frequencies leads to the additional wrong suggestion *lukken* ('succeed'), with trigram frequencies of 0, 108 and 49. For the next processing step (see Section 5.2), we also calculate the relative usage frequencies for the three context positions, in this case 77871/828=94.05, 22956/35=655.89 and 8387/3=2795.67. The median value, here 655.89, we deem the most indicative of the form's desirability as an alternative and is used as described below (under the name *median relative frequency*, Section 5.2).

17. However, we limit this examination to 100,000 instances, so that for 2,868 very frequent types a random sample of occurrences was used instead of the full set.
18. In this, we follow much the same strategy as is used in some systems designed for spelling correction for Dutch, such as Valkuil (http://valkuil.net; van den Bosch (personal communication)) and TICCL (Reynaert 2005).
19. Including trigrams which extend beyond the start or end of the tweet, in which case the trigram was padded with tokens consisting of a hash character.

For reasons of efficiency, steps a) and b) are executed in reverse order: better forms are suggested on the basis of context frequencies and these forms are then checked for edit distance.

5.2 Globally better alternatives

We now need to choose the best variant when considering all occurrences together. This in fact needs two decisions. The system not only needs to choose the best of all suggested variants, but also whether this best variant should be preferred to the original form. For both decisions, we might consider taking into account many features. We are currently using the following: the number of occurrences in the corpus, the number of different contexts (here one token left and right), the number of occurrences for which no better alternative was suggested, the number of occurrences for which the best alternative was suggested, the quotient of the latter two, the number of contexts for which no better alternative was suggested, the number of contexts for which the best alternative was suggested, the quotient of the latter two, the edit distance, the average over all occurrences where the best alternative was suggested of the median relative frequency (see above, Section 5.1), and the highest value for the median relative frequency over these occurrences.

We have experimented with several manually designed formulae to combine these features in order to decide whether to keep the original form as normal form or to choose the best alternative. However, as the interaction of the features proved rather complex, we decided to use a machine learner in the experiment at hand. We chose WPDV (van Halteren 2000) as this will automatically also take into account all combinations of features. A disadvantage of WPDV is that it can only handle nominal values, so that we had to discretize the values. For every feature, we mapped the continuous value to one of seven nominal values in such a way that the number of cases for each of the seven nominal values is approximately the same. We then took a sample of all word form types for which a choice needed to be made, attempting to include sufficient examples for all feature values. We annotated the resulting sample of 581 cases by hand and trained the hyperparameters by hill-climbing (cf. van Halteren 2000). The final classifier reached an accuracy of about 86% when applied in leave-one-out mode on the training set.

If the best variant selected is different from the original type, we apply one more operation. If the selected variant has also been assigned a better alternative variant, we use the latter variant instead. This chaining process is repeated until no more changes are made. For example, the form *gaaaaaaaaaa* is first mapped to *gaaaaa*, but then chained to *gaaa*, *gaa* and finally *ga* ('go').

6. Evaluation

We evaluated system behaviour for only five of our seven word form type classes. We did not evaluate straattaal as there is not really a consensus spelling and we are too unfamiliar with the subtleties in meaning to be able to judge a) when two types are variants of each other or not, and b) whether a straattaal word maps sensibly to a similarly spelled traditional Dutch word. We also did not evaluate foreign words, as our current algorithm is targeted at Dutch only. For the other five classes, we now list the results.

6.1 Standard forms

We start our evaluation with the standard forms. Here, we distinguish

- The form found in the tweets can itself function as the normal form.[20] The system suggests the same.

20. Remember that we are working with form types, and that this sentence should not be interpreted as meaning that the form should be given this normal form in a specific context, but is the normal form which is to be used most often.

Table 5: Evaluation of the system's single best normal form suggestion for standard forms. The string left of the slash indicates the authors' opinion and the one to the right of the slash the system output. SELF means that the form as used in the tweets is itself the normal form; ALT that another form is the normal form. In case of another form, the system may agree (ALT+) or disagree (ALT-) as to which alternative form is the best normal form.

Frequency band	SELF/SELF	ALT/ALT+	SELF/ALT	ALT/SELF	ALT/ALT-	Total
> 1/1,000	135	2	-	-	-	137
> 1/10,000	594	14	22	10	4	644
> 1/100,000	728	58	24	10	4	824
> 1/1,000,000	603	101	27	13	4	748
> 1/10,000,000	434	139	18	58	11	660
Total	2,494	314	91	91	23	3,013

- There is a better alternative spelling to be used as normal form. The system suggests the same alternative.
- The form found in the tweets can itself function as the normal form. The system suggests an alternative form.
- There is a better alternative spelling to be used as normal form. The system suggests using the form present in the tweets.
- There is a better alternative spelling to be used as normal form. The system suggests another alternative form.

The numbers for each evaluation class in each frequency band are shown in Table 5. Since the normal form is by definition the most frequent form in each variant group, it can be expected that the more frequent forms have a higher probability of being normal forms. This also becomes visible in the table, where we see that the percentage of forms that need alternatives increases as the frequency decreases. However, most of the types already turn out to be normal forms and the number of types needing alternatives is far lower than one might expect after experiencing Twitter text. Looking at agreement in general, we see an accuracy of about 93%, which remains practically stable in the three center frequency bands and only drops (to about 87%) in the last band.

If we inspect the incorrect suggestions in more detail, we see a number of ways in which the algorithm can err. When it incorrectly suggests alternatives, we see that it overmaps to different variant groups. Often, the overmapping jump is small, e.g. *grot-grote* ('cave'-'big') or *comma-coma*. However, we also see quite large differences, especially where there is semantic similarity and one form is markedly less frequent, such as in *honkbalwedstrijd-voetbalwedstrijd* ('baseball match' - 'soccer match', soccer being much more popular than baseball in the Netherlands). In two cases, the system at first stays close to the correct meaning, but then goes astray in the suggestion chaining: *mama-ma-man* and *pps-ps-pas* ('postscript'-'recently'). A quite common error (13 cases) is the confusion of verb forms. This can be between present and past tense (*werkte-werkt* ('worked' - 'works'); *tweette-tweet* ('tweeted' - 'tweets')), participle and infinitive (*genoten-genieten* ('enjoyed' - 'enjoy'); *rennend-rennen* ('running' - 'run')), but often also demonstrates the inability of many Dutch people to properly choose between *d, t* and *dt* in the formation of past participles and present tense (*gebeurd-gebeurt* ('happened' - 'happens'); *verandert-veranderd* ('changes' - 'changed'); *vindt-vind* ('finds' - 'find'); *wordt-word* ('becomes' - 'become')). We hypothesize that there must be relatively many incorrectly spelled forms in the corpus, especially for the latter two examples. If there were not, there are far too many correct examples and the system should be unable to learn the incorrect mappings.

Table 6: Evaluation of the system's single best normal form suggestion for jargon forms. The string left of the slash indicates the authors' opinion and the one to the right of the slash the system output. SELF means that the form as used in the tweets is itself the normal form; given the nature of jargon, we ourselves marked all forms as SELF (see text). The system may also suggest another form. In such cases we may judge this suggestion as acceptable (ALT+) or not (ALT-).

Frequency band	SELF/SELF	SELF/ALT+	SELF/ALT-	Total
> 1/1,000	3	1	-	4
> 1/10,000	9	3	-	12
> 1/100,000	16	9	3	28
> 1/1,000,000	11	3	5	19
> 1/10,000,000	5	1	1	7
Total	44	17	9	70

When the system does not manage to find the alternative form selected by us, we see that it cannot identify a present link and therefore undermaps. This can be surprising, e.g. when the system is unable to connect the non-existing form *erq* to the expected alternative *erg* ('very'), but sometimes also understandable, e.g. when it does not link *mechien* to *misschien* ('maybe'). Here too, we see various special classes. Above, we already mentioned that for some forms, different spelling variants occur in near equal frequencies; it should not come as a surprise that some of these will then lead the system to come up with erroneous suggestions, e.g. *papagaaien* instead of *papegaaien* ('parrots') or *nopes* instead of *noppes* ('nothing'). Similarly, many reduced forms are not corrected to their full form, such as *goeie* to *goede* ('good'), *tuurlijk* to *natuurlijk* ('of course') or *'m* to *hem* ('him'). For dialectal forms, e.g. not linking *woar-waar* ('where') or *skoon-schoon* ('clean', 'pretty'), it may be that the link is not made because the context is often also in dialectal spelling. The final class of errors, suggesting alternative forms other than the alternative we selected, is most often caused by following links that appear to be more likely than the needed ones. The form *scheiten*, for example, should be corrected to *schijten* ('poop'), with the other (and here correct) spelling for the same sound. However, the system prefers *schieten* ('shoot'), which is reached by transposition of the *e* and the *i*. Apparently the two words occur often enough in similar contexts that the mistake can be made. Even more surprising given the context restrictions is *hbet*. Here, there is a transposition and the system should suggest *hebt* ('have'). However, deleting the *h* and inserting the *n* yields the also very frequent form *bent* ('are') which the system prefers.

6.2 Jargon

One would assume that Twitter jargon forms can in all cases be used as their own normal form. The system, however, suggests alternatives for 26 out of 70 forms (Table 6). On inspection, 17 of these are not really wrong. Most often, the jargon has been formed by removing vowels from the original word and the system manages to reconstruct the original. Examples are *mn-mijn* ('my'), *gwn-gewoon* ('just') and *lkkr-lekker* ('good'). Special cases are *tg-toch* ('after all'), where the system does not only reinsert the vowels, but also corrects the spelling of the final consonant, and *vnv-vanaaf* ('this evening'), where the system suggests another jargon form for the correct sense instead of the full word *vanavond*. If we count all these cases as correct, we arrive at an accuracy of about 87% for Twitter jargon. In the remaining, fully erroneous cases, the system is overmapping, from mildly (*pww-pw* (*proefwerk* = 'exam' - *proefwerkweek* = 'exam week'), *wss-was* (*waarschijnlijk* = 'probably' - 'was')) to wildly (*hdd-tijd* ('hard disk' - 'time'); *ft-heeft* ('ft' - 'has')). A partial match is *zh-huis*, where *zh* is jargon for *ziekenhuis* ('hospital'). Also interesting is *rrt-rt* ('reretweet'-'retweet'),

Table 7: Evaluation of the system's single best normal form suggestion for names. The header for every class indicates the way in which the system agrees or disagrees with the authors (see text).

Frequency band	Agreement	Same name	Other name	Not a name	Total
> 1/1,000	-	-	-	-	-
> 1/10,000	11	-	-	-	11
> 1/100,000	68	1	-	4	73
> 1/1,000,000	119	-	3	3	125
> 1/10,000,000	173	10	6	3	192
Total	371	11	9	10	401

Table 8: Evaluation of the system's single best normal form suggestion for interjections. The header for every class indicates the way in which the system agrees or disagrees with the authors (see text).

Frequency band	Same cluster	Other interjection	Not an interjection	Total
> 1/1,000	9	-	-	9
> 1/10,000	39	5	-	44
> 1/100,000	37	11	2	50
> 1/1,000,000	39	9	2	50
> 1/10,000,000	35	15	4	54
Total	159	40	8	207

as something similar happens for the full verb *retweeten* ('retweet'), for which the system suggests *tweeten* ('tweet').

6.3 Names

For proper names, there is always a single official spelling. However, it is very often impossible to know this correct spelling without actually knowing the people (or other referred entities) in question. Given the nature of the text material in question, we are unable to identify these referred entities and therefore to evaluate reliably. As an alternative, we distinguish between a) the form suggested by the system and the form encountered are the same, b) the form suggested is a clear variant of the same name, c) the form suggested is clearly another name and d) the form suggested is not even a name. The resulting counts are shown in Table 7. If we count the first two columns as correct, the system's accuracy is about 95%. The system proves to be reluctant to suggest alternatives for names. In 11 of only 30 cases where the system does suggest an alternative to the encountered form, another spelling of the same name is suggested, e.g. *zarah-sarah* or *voss-vos*. A further 9 suggestions concern recognizably different names, e.g. *thies-dennis* and *tomtom-tom*. For the 10 suggestions, a non-name is suggested, e.g. *hov-of* ('or') and *stan-staan* ('stand'). For the six names where we decided to suggest a better alternative, the system was able to follow us for four of them (*bredaa-breda*, *sneeek-sneek*, *turkijeee-turkije*, *jaxxie-jaxie*). *Zitterd* (dialect form for the place name *Sittard*) and *coronas* (without the apostrophe before the *s*) were not linked to their officially spelled form.

6.4 Interjections

With interjections, one should rather evaluate the normal form in terms of the expressed emotion. Although this is clearly an interesting topic for a future study, we will here not attempt to identify

Table 9: Evaluation of the system's single best normal form suggestion for forms with tokenization problems. The string left of the slash indicates the authors' opinion and the one to the right of the slash the system output. SELF means that the form as used in the tweets is itself the normal form; ALT that that another form is the normal form. In case of another form, the system may agree (ALT+) or disagree (ALT-) as to which alternative form is the best normal form.

Frequency band	SELF/SELF	ALT/ALT+	SELF/ALT	ALT/SELF	ALT/ALT-	Total
> 1/1,000	-	-	-	-	-	-
> 1/10,000	7	1	-	-	-	8
> 1/100,000	20	2	1	1	-	24
> 1/1,000,000	21	3	-	2	-	26
> 1/10,000,000	27	7	3	5	-	42
Total	75	13	4	8	-	100

the emotion expressed, but merely distinguish between a) an interjection clearly falling in the same variant group, b) another interjection, and c) non-interjection. The resulting counts are shown in Table 8. With an accuracy of only about 77%, the performance of the system appears to be disappointing. However, 24 of the 48 errors are mismappings of forms of *ah*, *aha*, *whaha* and *wuhaha* to *haha*, 18 of which because of suggestion chaining. Once proper decisions have been made as to what clusters are desired for interjections, it is likely that the more frequent clusters (including most of the ones involved here) will be handled by regular expressions rather than machine-learned variation and such errors will no longer occur.

6.5 Forms with tokenization problems

The final class is the group with tokenization problems. In principle, one would expect that context would have to be handled in a different way for these forms. However, as most of the forms in question are clitics, the system can still process them quite adequately, with an accuracy of 88% (Table 9). In three of the four cases where the system incorrectly suggests an alternative, the suggested form is actually one of the two tokens that were merged together: *endan-dan* ('and then' - 'then'), *geeneen-geen* ('no one' - 'none') and *liefhe-lief* ('sweet, no' - 'sweet'). The final incorrectly suggested alternative and the 8 missed suggestions for alternatives are much like the errors that are made for the standard forms, where the system is unable to link to a mapped form (8 cases) or heads in the wrong direction (1 case).

7. Discussion and Future Work

In our experiment, we have tested whether our estimations of edit distance and of contextual similarity can be used to identify normal forms for word form types out of context. For most classes of form types, accuracies were around 90%, which is encouraging but also shows that improvements are still needed. However, our experiment has also led us to identify already quite a number of improvements which can be applied.

Our Vitershtein algorithm is working adequately in principle, but several aspects need to be tuned. For some, tuning was already intended. For example, the transition costs should be patterned on actually observed variation rather than on our initial intuitions. This may have to be done in an iterative learning process, each time learning variation patterns on the basis of variant groups formed on the basis of the costs in the previous cycle. We also want to experiment with transition costs that are differentiated as to the position in the respective strings. An example is the low cost of *n*-deletion, which should probably be limited to the end of the string (or the end of parts of

compounds). We have also observed several recurring variation patterns which have so far not been included in the algorithm, such as the ending *-uh* replacing *-en*. Finally, once the algorithm behaves well on alphabetic strings, we will extend it also to alphanumeric spellings like *w8* (*wacht* = 'wait').

The context window of two tokens left and right also appears to be sufficient as most cases are handled appropriately, but there are clearly cases of overmapping where more (or more intelligent) context is needed. Also, in future work, we will have to identify normal forms in context, which means the context in a single tweet. For that task, we will need to re-experiment. After starting with the same window as used in this paper, we may investigate increasing the size of the window when the immediate context is insufficiently disambiguating or even including all words in the tweet (as a bag of words) if we have to choose between two normal forms of the same word class where the choice can be made on the basis of the forms' meaning. In cases where it is hard to choose between forms of different word classes, we might investigate the possibility of (at least local) syntactic analysis.

A point requiring special attention with regard to context is the tokenization. Our algorithms only work properly for genuinely separate words and for many clitics. In other cases where words run together or are incorrectly separated, both variant group identification and context similarity estimation become problematic. Some kind of retokenization step seems appropriate (cf. van Halteren and Rem (submitted)), but we will need to investigate if this step is best inserted before or after the suggestion of normal forms for the tokens, or if the two processes are best interleaved.

All in all, our first step on this new path was encouraging. We have seen that (a type of) normal form can be identified in most cases, and we have identified several improvements to the estimations described in this paper which will make the process even more effective. Our primary conclusion is therefore that, after applying the improvements, these estimations will be a valuable aid in our further progress along the road on which we set out.

References

Levenshtein, V.I. (1966), Binary codes capable of correcting deletions, insertions, and reversals, *Soviet Physics Doklady* **10**, pp. 707–710.

Reynaert, M.W.C. (2005), *Text-Induced Spelling Correction*, PhD thesis, University of Tilburg.

Reynaert, M.W.C. (2006), Corpus-induced corpus cleanup, *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-06), Genoa, Italy, 2006.*

Stehouwer, H. and M. van Zaanen (2010), Using suffix arrays as language models: Scaling the n-gram, *Proceedings of the 22nd Benelux Conference on Artificial Intelligence (BNAIC-2010), Luxembourg.*

Tagg, C. (2009), *A corpus linguistic study of SMS text messaging*, PhD thesis, University of Birmingham.

Tjong Kim Sang, E. (2011), Het gebruik van Twitter voor Taalkundig Onderzoek, *TABU: Bulletin voor Taalwetenschap* **39** (1/2), pp. 62–72.

van Berkel, B. and K. De Smedt (1988), Triphone analysis: A combined method for the correction of orthographical and typographical errors, *ANLP*, pp. 77–83.

van de Cruys, T. (2010), *Mining for meaning: the extraction of lexico-semantic knowledge from text*, PhD thesis, University of Groningen.

van de Velde, H. and R. van Hout (2003), De deletie van de slot-n, *Nederlandse taalkunde* **8** (2), pp. 93–114.

van de Velde, H., M. Gerritsen, and R. van Hout (1996), The devoicing of fricatives in standard Dutch. A real time study based on radio recordings, *Language Variation and Change* **8** (2), pp. 149–176.

van Halteren, H. (2000), A default first order family weight determination procedure for WPDV models, *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning*, pp. 119–122.

van Halteren, H. and M. Rem (Submitted), Dealing with orthographic variation in a tagger-lemmatizer for 14th century Dutch charters.

Vandekerckhove, R. and J. Nobels (2010), Code eclecticism: Linguistic variation and code alternation in the chat language of Flemish teenagers, *Journal of Sociolinguistics* **14** (5 (November, 2010)), pp. 657–677.

Viterbi, A.J. (1967), Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Transactions on Information Theory* **13** (2), pp. 260–269.