Using idiolects and sociolects to improve word prediction

Wessel Stoop
Centre for Language and Speech Technology
Radboud University Nijmegen
w.stoop@let.ru.nl

Antal van den Bosch
Centre for Language Studies
Radboud University Nijmegen
a.vandenbosch@let.ru.nl

Abstract

In this paper the word prediction system *Soothsayer*[1] is described. This system predicts what a user is going to write as he is keying it in. The main innovation of Soothsayer is that it not only uses *idiolects*, the language of one individual person, as its source of knowledge, but also *sociolects*, the language of the social circle around that person. We use Twitter for data collection and experimentation. The *idiolect* models are based on individual Twitter feeds, the *sociolect* models are based on the tweets of a particular person and the tweets of the people he often communicates with. The idea behind this is that people who often communicate start to talk alike; therefore the language of the friends of person x can be helpful in trying to predict what person x is going to say. This approach achieved the best results. For a number of users, more than 50% of the keystrokes could have been saved if they had used Soothsayer.

1 Introduction

The main aim of the study presented here is to show that the concepts of *idiolect* and *sociolect*, the language of one person and his or her social circle, can be used to improve *word prediction*, the task of predicting what a user is going to type, as he is typing. Word prediction technology reduces the number of keystrokes we have to make, thus saving time and preventing mistakes. With the rise of smartphones word prediction has become widely known and used. Preceding this popularization, word prediction systems were already developed up to three decades ago to assist people with speech and motor disabilities, like cerebral palsy or hemiplexia. By using a device equipped with word prediction technology, they can increase their communication rate considerably (Garay-Vitoria and Abascal, 2006). Indeed, most studies before the year 2000, when mobile phones were not widely used yet, targeted the disabled user group - Copestake (1997) even reports building a system for one individual user. More recent work targets a wider audience, but in this paper we return to the idea of using an individual's own language to train individualized models.

The concept of an idiolect, the language of a single person, is well-known, but rarely ever modelled or in some other way operationalized (Mollin, 2009; Barlow, 2010; Louwerse, 2004). Almost every claim in the field of linguistics concerns language as a whole; whether the subject of investigation is a particular syntactic construction, phonological variable, or some other linguistic phenomenon, the results are always supposed to hold for an entire language variety. According to Mollin (2009) idiolects are a 'neglected area in corpus linguistics', and Barlow (2010) states that the term 'is distinguished by the fact that there is probably no other linguistic term in which there is such a large gap between the familiarity of the concept and lack of empirical data on the phenomenon.' This is remarkable, since 'idiolects are the only kind of language we can collect data on', as Haugen (1972) points out; a language variety essentially is a collection of idiolects.

Word prediction systems typically operate with an algorithm and a language model, as the overview of related work in Section 2 will show. Language models are created from training material, typically a large collection of text. Section 3 introduces our algorithm step by step. The resulting best-performing algorithm is used in Section

[1]The system is available as an interactive demo at http://soothsayer.cls.ru.nl/ and its source code is publicly available at https://github.com/woseseltops/soothsayer

Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 318–327,
Gothenburg, Sweden, April 26-30 2014. ©2014 Association for Computational Linguistics

4, in which we investigate which language model should be used together with this algorithm. We start with the notion of an individual's idiolect in Section 4.1, and expand this by using the language of the people this individual communicates with, in Section 4.2. In Section 5 we offer our conclusions and formulate points for further research.

2 Related work

An early solution for word prediction was to use word frequency lists (Swiffin et al., 1985). Although it is possible to wait until a word unicity point has been reached (the point in a word where there there is no other word with the same prefix), more keystrokes may be saved if the prediction can be done before the unicity point. After this first data-driven improvement, numerous authors have shown that taking the contextof previously entered words into account improves prediction accuracy further. A simple approach to implementing context-sensitivity is applying the frequency list technique to word n-grams (Hunnicutt, 1987); in a string of work other statistical language modeling approaches have been proposed (Lesher et al., 1999; Langlais et al., 2000; Garay-Vitoria and Abascal, 2006; Tanaka-Ishii, 2007; Van den Bosch and Bogers, 2008).

The accuracy of a context-sensitive system largely depends on how often a similar context is available in the training material; the amount of training data will be an important factor for the system's success. A key publication by Lesher *et al.* (1999) indeed shows that the accuracy of a context-sensitive word prediction system is related to how much training material is provided. On the other hand, once most of the frequent combinations are covered, it takes more and more training material to improve the results a little bit. Van den Bosch (2011) demonstrates that the relation between the amount of training data and word completion performance is roughly log-linear. For instance, when going from 100 to 1,000 words in the training material, roughly 6% more keystrokes could be saved (from 15% to 21% keystrokes saved), while the same is true for the step from 1,000,000 to 10,000,000 words (from 40% to 46%).

A large portion of the work on word prediction includes linguistic knowledge in some way, for example by also training the system which PoS-tags are likely to follow each other, and using that to limit the pool of suggestions (Carlberger et al., 1997; Fazly and Hirst, 2003; Copestake, 1997; Matiasek et al., 2002; Garay-Vitoria and Gonzalez-Abascal, 1997). Interestingly, most authors conclude that including linguistic knowledge improves the results, but only slightly (Garay-Vitoria and Gonzalez-Abascal, 1997; Fazly and Hirst, 2003). Fazly and Hirst (2003) note that adding explicit linguistic knowledge 'might not be considered worth the considerable extra cost that it requires'. In the current study we have not used any explicit linguistic knowledge, thus making our system language-independent.

There have also been more successful optimizations of word completion systems. One is to use training material from the same domain. Verberne *et al.* (2012) show that trying to predict Wikipedia text, tweets, transcriptions of conversational speech and Frequently Asked Questions all worked best when using texts from the same type. As a second optimization, building on the idea of cache language models (Goodman, 2001), Van den Bosch (2011) proposes to make a word completion system learn about register and topic on the fly with a *recency buffer*. This buffer stores the n latest words; if a word the user is keying in matches one of the words in the recency buffer, this word is suggested instead of what the system would actually have suggested. The idea behind this is that if the user is writing about, for example, traveling, words like 'go', 'hotel', and 'see' are likely to be in the buffer and thus could be suggested quickly. In other words, the systems learns about the user and the topic on the fly.

Although both approaches help to increase the number of keystrokes saved, they also have downsides: for the system by Verberne *et al.* (2012) training texts in the same genre are needed, which might not be available, whereas the system by Van den Bosch (2011) ignores context information that it should weigh more intelligently. For example, while a context-sensitive text-prediction system will probably be able to predict *to* for the sentence *they were going t...*, the one with the recency buffer will predict *they*.

3 System description

Soothsayer predicts the next word the user is going to type, or the rest of the word in case the user already starting typing something. To do this, it works with a set of independent word pre-

diction modules. Modules can be either context-insensitive or context-sensitive, and use one language model. We will work with two language models, one based on a large collection of texts sampling from many different authors, the 'general language model', and one based on a set of texts written by an individual, the 'idiolect'. We thus have four possible modules:

Module	Type	Model
1	Context-sensitive	idiolect
2	Context-sensitive	general language model
3	Context-insensitive	idiolect
4	Context-insensitive	general language model

Table 1: Four possible modules: combinations of type and language model

Modules can be concatenated in such a way that a second module takes over once the first modules no longer has predictions, a third module takes over once the second one no longer has predictions, etc. In future work, interpolation of the predictions of these modules should be investigated.

3.1 Context-insensitive modules

Context-insensitive modules only use information of the word the user is currently keying in. In sentence 1, for example, only the c will be used for prediction.

(1) I ate too much c

This means that a prediction like *communication* is fully possible, despite the context. This also means that at the beginning of each new word no prediction will be available, because the module has no material to work with. Despite these limitations, context-insensitive modules can already save a lot of keystrokes, because the first few letters of a word impose strong limitations on what letters can possibly follow, and some words have early unicity points. Predictions are done by going through a frequency list, so the most frequent (and thus more likely to occur again) words are considered first. Once a word is encountered that matches what has been keyed in so far, it is suggested.

3.2 Context-sensitive modules

Context-sensitive modules make use of the words that came before the current word to limit what words are predicted. Soothsayer approaches word prediction as a classification task, where the three words in the left context of the word to be predicted are the features, and the word following this context is the class label to be predicted. This means that we have a separate class for every word that could possibly be predicted. Soothsayer uses the k-nearest neighbour classification method, which is insensitive to the number of classes to be predicted. k-nearest neighbour classification (henceforth *KNN*) means that the class is determined on the basis of similar cases in a training corpus. How many cases are taken into consideration, k, can be determined beforehand. The similarity between a new instance and memorized instances is determined using a simularity function. A classic implementation of *KNN* suited for the type of symbolic features we have, the IB1-algorithm (Aha et al., 1991), simply counts how many features overlap. However, the IB1 algorithm generally is too slow to be used in practical applications, ours included. We adopt IGTree[2] (Daelemans et al., 1997), an approximation of IB1 that does not require a comparison of the complete context.

IGTree calculates which features contain most information about the class labels using the Information Gain or Gain Ratio metrics, orders the features from most informative to least informative, and compresses the training set in a decision tree. Classification of a new context reduces to making a small number of decisions (a maximum of three, because we have three features), instead of comparing a new context to thousands to millions of contexts. If we ask IGTree to classify unseen input, the algorithm may not come to a unique decision. Soothsayer asks the algorithm to return everything it considers a possibility at the deepest node it reached while traversing the decision tree. Analogous to the manner in which the context-insensitive module generates frequency-ranked lists of possible completions, IGTree will produce a frequency-ranked list of possible completions it found at this deepest node in the tree. Soothsayer then accepts the single (or three) top-ranking suggestion(s).

3.3 Evaluating the system

There is an ongoing discussion in the literature on what is the best way to evaluate word pre-

[2]IGTree is implemented in the TiMBL software package, `http://ilk.uvt.nl/timbl`

diction systems. A straightforward evaluation might be to calculate the percentage of correctly predicted words (the so-called *hit-ratio*), but as Garay-Vitoria and Abascal (2006) note, this is not enough: a system that has 100% of the words correct, but only gives this prediction at the last letter of every word saves very few keystrokes. A more natural way might be to test with real humans, and measure how much time they save when using the system (Carlberger et al., 1997; Koester and Levine, 1998; Garay-Vitoria and Abascal, 2006). However, this is a costly and time-consuming task, as the participants will need a considerable amount of time to get used to the system. Therefore, we will evaluate Soothsayer by simulating somebody keying in a text, and counting how many keystrokes this virtual user does not have to do when using the system.

However, even when using simulations, there are multiple ways to evaluate the system. One possibility is to provide the user with a list of the n most likely predictions (Lesher et al., 1999; Fazly and Hirst, 2003). This approach has the advantage that it results in high percentages of keystrokes saved - in particular when n is set to a high value, because this means the system can do multiple guesses at once, while only one has to be correct. As Van den Bosch (2011) notes, however, this also has important downsides:

> [I]n many devices and circumstances it is inefficient or impossible to present [...] suggestion lists. Inspecting a list of suggestions also poses a larger cognitive load than checking a single suggestion, and furthermore it is unclear how scrolling, browsing and selecting items from this list should be counted in terms of keystrokes.

For this reason, we calculate the number of keystrokes that could have been saved when the user was presented only one prediction at a time. Predictions can be accepted with the space key. Because this is sometimes problematic (for instance, if the user wanted to type *sun*, but Soothsayer predicts *sunday*, hitting space would lead to the wrong word), a rejection is also calculated as a keystroke. The number of keystrokes that can be saved if the word prediction system works this way will be called **Classical Keystrokes Saved** (CKS) in the remainder of this paper. Please note that CKS assumes that the user always accepts a prediction immediately when it is available, which might not always be the case in reality.

On the other hand, current popular smartphone applications suggest this approach might be too strict. The popular smartphone application SwiftKey[3] always shows the user three predictions, which seem to be (1) what the user has keyed in so far, (2) the most likely prediction and (3) the second most likely prediction. In case the user has not yet started typing the next word, option (1) is replaced by the third most likely prediction. The percentage of keystrokes that can be saved when two (and sometimes three) predictions were shown will be referred to as **SwiftKey Keystrokes Saved** (SKKS). This percentage will mostly be higher than CKS.

3.4 Other considerations

Context-sensitive before context-insensitive The context-sensitive module learns about which words in the training texts typically follow each other, and thus is potentially powerful when it comes to the more frequent, fixed combinations of words and words that often occur in each other's context, but is not useful with words that are also frequent, but were not used earlier in this context. The context-insensitive module, on the other hand, can predict any word, as long as it has been used before, but knows nothing about fixed combinations. In other words, the modules complement each other. Based on the fact that context-sensitive modules have been reported as scoring better than context-insensitive modules in direct comparisons in controlled experiments (Lesher et al., 1999), we rank context-sensitive modules before context-insensitive ones in all studies reported here. The context-insensitive module trained on idiolects precedes the final context-insensitive module trained on a general language corpus; this order reflects the order also found in the context-sensitive modules described in more detail in the next section.

Attenuation IGTree is fast in classification, but with tens of millions of training instances it becomes too slow for real-time use, where fast typists may reach as many as twenty keystrokes per second. To alleviate this issue we use a (simplified version of a) solution from the field of syntactic parsing called *attenuation* (Eisner, 1996). All

[3] http://www.swiftkey.net/

words in the training material that occur less often than a particular threshold are replaced by a dummy value. Replacing all low-frequent words by one dummy value makes the IGTree considerably smaller and thus faster to traverse during classification. In a pilot experiment an attenuation threshold of 3 turned out to be the most desirable: it leads to the largest increase in speed (from 28 classifications per second to 89) without any measurable decrease in prediction accuracy. For this reason, an attenuation threshold of 3 was used throughout the study.

Handling morphology Some aspects of morphology are inherently problematic for word completion, in particular compounding, inflections, and suffixes. For example, imagine a user has already written sentence 2, and wants to write the word *cookies*:

(2) I would really like the c

If in the training material the word *cookie* was more frequent, Soothsayer will suggest that instead of *cookies*. Normally, when a prediction is wrong, the algorithm will find out because the user keys in another letter (so the predicted word no longer matches what the user is typing), but that technique will not work here. For words that only differ in their suffix, the point of difference is at the end of the word, when there is nothing left to predict. Even if the correct word is the second most likely prediction, this will not be suggested, because Soothsayer has no reason to switch prediction.

However, there is a clue Soothsayer could use: normally, when a prediction is right, the user will accept it, instead of going on writing. He might not accept it immediately (typing often goes faster than mentally processing predictions), but once the user has not accepted a prediction for more than two or three keystrokes in a row, it gets more and more likely the user keeps on typing because the prediction is wrong. In that case, the second most likely prediction could be displayed, which in many cases will be the word with the second most likely suffix. We use this *early prediction switching* method throughout our experiments.

Recency As Church (2000) showed, the probability that a word recurs twice in a short stretch of text is far higher than its frequency in language would suggests, which is mainly related to the word's topicality. Whereas knowledge about topics could be covered by training and testing within the same coherent set of texts (e.g. all written by a single person), the aforementioned recency buffer by definition uses more recent text (that is, material from the same text), and might this way be able to do more accurate predictions. We implemented a buffer that remembers the n most recent words, and suggests the most recent one that matches with the word that is currently being keyed in. Following Van den Bosch (2011) we set n to 300. If no word matches, the next module will take over. In our experiments we have tested the insertion of the recency buffer module after the context-sensitive modules and before the context-insensitive modules.

4 The model: idiolects and sociolects

4.1 Idiolects

In this experiment the power of idiolects will be investigated by training and testing an array of systems on one hundred different idiolects of individuals. For this, the micro-blogging service Twitter[4] is used. Twitter is a micro-blogging service where each user can submit status updates known as tweets, which consist of 140 characters or less. Using the Twitter API, all tweets of a manually created seed set of 40 Dutch Twitter users were retrieved from January until June 2013. Retweets, messages these authors did not produce themselves, were excluded. These seed users were the starting point of an iterative expansion of the set of crawled Twitter uses by following mentions of other users (indicated with the syntax *'@username'*). The goal of this expansion was to find as much active Twitter users as possible for the system to follow, and to capture the network of these users. The set was extended with two users every 30 minutes with the following procedure:

- From the set of users mentioned in tweets already harvested and not yet tracked, the most frequently mentioned user is selected. This ensures that the new person communicates with at least one of the persons the system is already following.
- From the set of users mentioned by more than a single person already being tracked, the most frequently mentioned user is selected. This ensures the new person is well

[4] http://www.twitter.com

connected to the people the system is already following.

The system limits itself to Dutch tweets using a conservative Dutch word frequency list containing highly frequent Dutch words that have no counterpart in other languages.

Concerning the relation between number of tweets and Twitter users, many scholars have noticed that it follows a Pareto-distribution (Heil and Piskorski, 2009; Asur and Huberman, 2010; Rui and Whinston, 2012). That is, a small part of the Twitter users produce a large part of the tweets. This distribution means that using all or a random selection of Twitter users is not likely to lead to good results, because for most users not much material is available. Therefore, only data from the 100 Twitter users for which the most material was harvested are used to build the idiolect models. Twitter accounts run by something other than an individual person (such as a company) were excluded manually. The number of words ranged from 61,098 words for the least active user of the 100 users to 167,685 words for the most active user. As a general language model, a random selection of blogs, emails and Wikipedia articles from the SoNaR corpus for written Dutch (Oostdijk et al., 2013) was made. These texts were chosen because they were believed to be neither very formal nor very informal, and fall in the same new-media category as Twitter messages. The general language corpus consisted of 55,212,868 words.

First, we compared the general language model against each user's idiolect, and tested on all 100 Twitter feeds of individual users. We then combined the two models (the general model acting as back-off for the idiolect model). These three setups were tested with and without a recency buffer module, resulting in six runs. For each of these runs, we tried to predict the 10% most recent material, and trained on the remaining 90% (for idiolects). Tables 2 and 3 list the results on these six runs measured in CKS and SKKS, respectively. We observe that using the idiolect model leads to more keystrokes saved than using the general model. We also see that using the general language model as a background model leads to more keystrokes saved than using the idiolect model alone. Using the recency buffer leads to more keystrokes saved, especially when it is used in addition to the general mode,

An ANOVA for repeated measures showed that there is a significant effect of the training material $F(2, 198) = 109.495, p < .001$ and whether the recency buffer was used $F(1, 99) = 469.648, p < .001$. Contrast analyses revealed that both the differences between the results of the general model and the idiolect model $F(1, 99) = 41.902, p < .001$ and the idiolect model and the idiolect model with the background model $F(1, 99) = 232.140, p < .001$ were significant.

The high standard deviations indicate a lot of variation. The substantial individual differences are illustrated in Figure 1, where the users are ordered from least to most material. Contrary to expectations, no correlation between amount of training material and the results could be detected (Pearson's correlation, $p = .763$); apparently, the individual factor is that much stronger, and Soothsayer performs much better for one than for the other. Using the overall best-performing module set-up, the set-up with the idiolect model, backed up by the general language model, *and* the recency buffer, the worst result is 21.8% CKS and 24.1% SKKS for user 90, and the best result is 51.3% CKS and 52.4% SKKS for user 97.

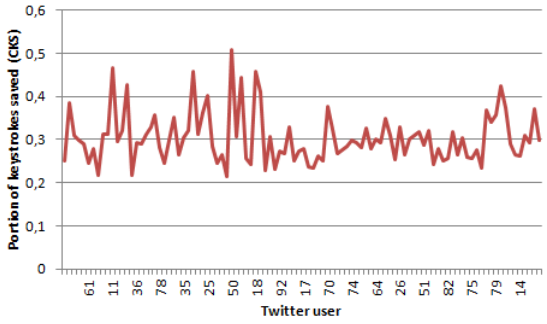

Figure 1: The proportion of keystrokes saved for individual Twitter users, ordered from by amount of tweets (from left to right: from least to most), when using the best-performing module set-up

The large amount of variation between individual Twitter users cannot easily be explained, with a few exceptions (for example, people with powerful idiolect models sometimes often repeated long words like *goedemorgen* 'good morning', *dankjewel* 'thank you', and *welterusten* 'sleep well'), but no clear patterns emerged. Trying to predict for which persons word prediction will go well and for which persons it will not might be an interesting topic for future research. It is a question that is related to the field of computational

Training material	Test material	Without recency buffer		With recency buffer	
		Mean	St. dev.	Mean	St. dev.
General	Twitter	14.4	5.1	23.2	5.2
Idiolect	Twitter	23.2	7.9	26.7	7.9
Idiolect + general	Twitter	26.4	6.2	29.7	6.4

Table 2: Mean percentage of keystrokes saved (**CKS**) and standard deviations for all module set-ups.

Training material	Test material	Without recency buffer		With recency buffer	
		Mean	St. dev.	Mean	St. dev.
General	Twitter	16.2	6.1	26	5.4
Idiolect	Twitter	24.8	8.3	27.9	7.2
Idiolect + general	Twitter	28.2	6.3	32.1	6.3

Table 3: Mean percentage of keystrokes saved (**SKKS**) and standard deviations for all module set-ups.

stylometry and in particular automatic authorship attribution, although authorship attribution is the exact opposite of the task described here (guessing the author on the basis of text instead of guessing the text on the basis of the author) (Bagavandas and Manimannan, 2008).

4.2 Social networks and language input

The findings by Lesher *et al.* (1999) suggest that more material leads to more keystrokes saved; this may also hold for idiolects. This material, however, might not be available, simply because not all people write or tweet that much. For a particular user x, what other sources of language do we have that might be similar to the idiolect of x? One of the more obvious answers might be the language of the people x often communicates with. The fact that people that are in some way related to each other speak alike using a 'group language' or a sociolect, is well established in sociolinguistics.

This approach of including the language of the people from a particular person's environment can also be viewed from a different perspective: so far, we have followed Mollin (2009) and Barlow(2010) in using only the *output* of speakers. This makes sense (since what comes out must have been inside), but can never be the full story. The sociolect model that will be constructed here can be seen as a feasible and rough approximation of recording everything a person reads or hears: by including the language of the socially related persons of person x, the system can have a rough idea of the kind of *input* person x gets.

On the basis of the data already collected for the idiolect experiments, sociolects were created by collecting all addressees mentioned with the *@addressee* syntax for each of the 100 Twitter users used in the previous experiment. For all addressees that were mentioned three times or more, it was checked if this addressee was in the dataset (which was almost always the case). If so, it was checked whether this addressee also mentioned the original Twitter user at least three times. If this was also the case, the system assumed the users speak to each other often enough to have their language adjusted to each other, and the tweets of this addressee were added to the sociolect of the original Twitter user. We thus end up with 100 sociolects built around the 100 most active Twitter users, all based on the tweets of a Twitter user and the tweets of the persons that this person communicated with at least six times (three times as writer, three times as reader).

The results of Verberne *et al.* (2012) would predict that adding tweets in general would lead to increases in the number of keystrokes saved, as this is using more texts from the same genre. To be sure that any improvements can be attributed to the fact that this is the language from friends, a control model will be built. While the sociolect model consists of the tweets of Twitter user x and the tweets of the friends of twitter user x, the control model consists of the tweets of Twitter user x and the tweets of random other Twitter users, and has approximately the same number of words.

For each of the 100 Twitter users, comparative runs are performed with the model created on the basis of the idiolect and the random Twitter users versus the sociolect model. The best performing module set-up from the previous experiments is used. The results are compared to the simulations with the idiolect model from the previous experi-

Training material	Test material	CKS		SKKS	
		Mean	St. dev.	Mean	St. dev.
Idiolect	Twitter feed	29.6	6.4	32.1	6.3
Control model	Twitter feed	31.2	6.3	33.9	6
Sociolect	Twitter feed	33.9	7.1	36.2	7.1

Table 4: Mean percentage of keystrokes saved when using an idiolect, a control model (consisting of an idiolect and random other Twitter feeds) and a sociolect.

Twitter user	Idiolect		Idiolect+random feeds		Sociolect	
	CKS	SKKS	CKS	SKKS	CKS	SKKS
24	31.2	36.3	34	36.4	31.6	34.3
49	27.2	29.1	26.2	29.7	24.6	27.2
71	27.5	30.2	34.2	35.8	30.8	32.9

Table 5: Percentage of keystrokes saved for 3 atypical Twitter users, using the the idiolect, control and sociolect models

ment. The results of the simulations are summarized in Table 4. We observe that adding more tweets to the idiolects leads to more keystrokes saved, and that the most keystrokes can be saved when using the tweets of the people the owner of the idiolect communicates with often.

An ANOVA for repeated measures showed that there is a significant effect for the training material $F(2, 198) = 69.466, p < .001$. Contrast analyses revealed that both the differences between the results of the idiolect model and the idiolect model and random feeds $F(1, 99) = 93.471, p < .001$ and the idiolect model and random feeds and the sociolect model $F(1, 99) = 61.871, p < .001$ are significant.

Again, the high standard deviations indicate notable variation among the individual results. Table 5 lists the deviating individual scores for three individual Twitter users. In these results we see an increase when random tweets are added, but a decrease when the tweets from their conversation partners are used. For user 24 and 49, the percentage of keystrokes saved when using the sociolect model is even lower than the idiolect model alone.

Using the best-performing module set-up in general, the set-up with the sociolect model, backed up by the general language model, *and* the recency buffer, the worst result is 21.3% CKS and 22% SKKS for user 90, and the best result is 56.2% CKS and 58.1% SKKS for user 38.

5 Conclusion

In this paper we presented the word prediction system *Soothsayer*. Testing the system we found that word prediction and idiolects are an effective combination; our results show that word prediction is best done with a combination of an idiolect-based context-sensitive system, backed up by a context-sensitive module equipped with a general language model. A recency buffer is a useful third module in the sequence. Our average best scores with these three modules are 29.7% keystrokes saved according to the strict (one-best) CKS metric, and 32.1% keystrokes saved according to the Swiftkey-inspired SKKS metric.

The fact that people speak like the people around them can also be useful to word prediction. When we approximate a sociolect by expanding a user's Twitter corpus by tweets from people this person communicates with, and retrain our first context-sensitive module with this data, average scores improve to 33.9% CKS and 36.2% SKKS.

What works well for one speaker, might not necessarily work for another, however. While we find significant advantages of idiolect-based and sociolect-based training, the variance among our 100 test users is substantial, and in individual cases idiolect-based training is not the best option. For other users the positive gains are substantially higher than the mean; the best result for a single user is 56.2% CKS and 58.1% SKKS.

In future research we aim to investigate methods that could predetermine which model and module order will work best for a user. Another set of open research questions concern the fact that we have not tested many of the system's settings. What would be the effects of predicting more words at the same time?

References

D. W. Aha, D. Kibler, and M. K. Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 6:37–66.

S. Asur and B. A. Huberman. 2010. Predicting the future with social media. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '10, pages 492–499, Washington, DC, USA. IEEE Computer Society.

M. Bagavandas and G. Manimannan. 2008. Style consistency and authorship attribution: A statistical investigation. *Journal of Quantitative Linguistics*, 15(1):100–110.

M. Barlow. 2010. Individual usage: a corpus-based study of idiolects. In *LAUD Conference, Landau*. http://auckland.academia.edu/MichaelBarlow.

A. Carlberger, J. Carlberger, T. Magnuson, S. Hunnicutt, S. E. Palazuelos Cagigas, and S. Aguilera Navarro. 1997. Profet, a new generation of word prediction: An evaluation study. In *ACL Workshop on Natural Language Processing for Communication Aids*, pages 23 –28.

K. W. Church. 2000. Empirical estimates of adaptation: The chance of two noriegas is closer to p=2 than p2. In *Proceedings of the 18th Conference on Computational Linguistics*, volume 1, pages 180–186.

A. Copestake. 1997. Augmented and alternative nlp techniques for augmented and alternative nlp techniques for augmentative and alternative communication. In *Proceedings of the ACL workshop on Natural Language Processing for Communication Aids*, pages 37–42.

W. Daelemans, A. van den Bosch, and A. Weijters. 1997. Igtree: using trees for compression and classiffication in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

J. Eisner. 1996. An empirical comparison of probability models for dependency grammar. In *Technical Report IRCS-96-11, Institute for Research in Cognitive Science*. University of Pennsylvania.

A. Fazly and G. Hirst. 2003. Testing the efficacy of part-of-speech information in word completion. In *Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods*, pages 9–16.

N. Garay-Vitoria and J. Abascal. 2006. Text prediction systems: a survey. *Univers. Access Inf. Soc.*, 4(3):188–203.

N. Garay-Vitoria and J. Gonzalez-Abascal. 1997. Intelligent word-prediction to enhance text input rate. In *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, pages 241–244.

J. Goodman. 2001. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434.

E. Haugen. 1972. From idiolect to language. In E. Scherabon Firchow, K. Grimstad, N. Hasselmo, and W. A. ONeil, editors, *Studies by Einar Haugen. Presented on the Occasion of his 65th Birthday*, pages 415–421. Mouton, The Hague/Paris.

B. Heil and M. Piskorski. 2009. New twitter research: Men follow men and nobody tweets. `http://blogs.hbr.org/cs/2009/06/new\_twitter\_research\_men\_follo.html`.

S. Hunnicutt. 1987. Input and output alternatives in word prediction. *STL-QPSR*, 28(2-3):015–029.

H. H. Koester and S. P. Levine. 1998. Model simulations of user performance with word prediction. *Augmentative Alternative Commun*, pages 25–35.

P. Langlais, G. Foster, and G. Lapalme. 2000. Transtype: a computer-aided translation typing system. In *Proceedings of the 2000 NAACL-ANLP Workshop on Embedded machine translation systems-Volume 5*, pages 46–51. Association for Computational Linguistics.

G. W. Lesher, B. J. Moulton, and D. J. Higginbotham. 1999. Effects of ngram order and training text size on word prediction. In *Proceedings of the Annual Conference of the RESNA*.

M. M. Louwerse. 2004. Semantic variation in idiolect and sociolect: Corpus linguistic evidence from literary texts. *Computers and the Humanities*, 38(2):207–221.

J. Matiasek, M. Baroni, and H. Trost. 2002. FASTY: A multi-lingual approach to text prediction. In *Computers Helping People With Special Needs*, pages 165–176. Springer Verlag, Berlin, Germany.

S. Mollin. 2009. I entirely understand is a blairism: The methodology of identifying idiolectal collocations. *Journal of Corpus Linguistics*, 14 (3):367–392.

N. Oostdijk, M. Reynaert, V. Hoste, and I. Schuurman. 2013. The construction of a 500-million-word reference corpus of contemporary written Dutch. In *Essential Speech and Language Technology for Dutch*, pages 219–247. Springer.

H. Rui and A. Whinston. 2012. Information or attention? an empirical study of user contribution on twitter. *Information Systems and e-Business Management*, 10(3):309–324.

A. L. Swiffin, J. A. Pickering, J. L. Arnott, and A. F. Newell. 1985. PAL: An effort efficient portable communication aid and keyboard emulator. In *Proceedings of the 8th annual coonference on Rehabilitation Technology, RESNA*, pages 197–199.

K. Tanaka-Ishii. 2007. Word-based Predictive Text Entry using Adaptive Language Models. *Natural Language Engineering*, 13(1):51–74.

A. Van den Bosch and T. Bogers. 2008. Efficient context-sensitive word completion for mobile devices. In *MobileHCI 2008: Proceedings of the 10th International Conference on Human-Computer Interaction with Mobile Devices and Services, IOP-MMI special track*, pages 465–470.

A. Van den Bosch. 2011. Effects of context and recency in scaled word completion. *Computational Linguistics in the Netherlands Journal*, 1:79–94.

S. Verberne, A. Van den Bosch, H. Strik, and L. Boves. 2012. The effect of domain and text type on text prediction quality. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France*, pages 561–569, New Brunswick, NJ. ACL.