Data-Driven Spelling Correction using Weighted Finite-State Methods

Miikka Silfverberg[a] **Pekka Kauppinen**[b] **Krister Lindén**[b]
Department of Modern Languages, University of Helsinki
[a]mpsilfve@iki.fi, [b]firstname.lastname@helsinki.fi

Abstract

This paper presents two systems for spelling correction formulated as a sequence labeling task. One of the systems is an unstructured classifier and the other one is structured. Both systems are implemented using weighted finite-state methods. The structured system delivers state-of-the-art results on the task of tweet normalization when compared with the recent AliSeTra system introduced by Eger et al. (2016) even though the system presented in the paper is simpler than AliSeTra because it does not include a model for input segmentation. In addition to experiments on tweet normalization, we present experiments on OCR post-processing using an Early Modern Finnish corpus of OCR processed newspaper text.

1 Introduction

Spelling correction is one of the most widely applied language technological utilities. The most obvious application of spelling correction is as a writer's aid. However, many natural language processing applications can also benefit from a spelling correction component. For example, many existing NLP systems are trained on newswire which tends to closely adhere to orthographical and grammatical norms. These systems may incur a substantial hit in performance when they are applied to noisy domains like social media. When spelling correction is applied as a preprocessing step, performance can be better. Digitization of documents is another domain where spelling correction is useful. Digitization often aims to transform physical documents into digital representations which support free text search. This requires the use of an optical character recognition (OCR) engine. Depending on the quality of the engine and source documents, this can succeed to varying degrees. Spelling correction can be applied as a post-processing step in order to improve quality.

input	A	m	.	c
output	A	nn	ε	e

Figure 1: Post-editing as sequence labeling. The input to the post-editor is "Am.c" and the correct output is "Anne". This representation corresponds to the 1-to-n alignment of Bisani and Ney (2008) because each input symbol is associated with a possibly empty sequence of outputs.

Spelling correction is an instance of the more general task of string-to-string translation. In spelling correction, the objective is to transform a possibly erroneous input string, for example a misspelling or OCR error, into a correct output string. Like many string-to-string translation tasks, spelling correction can be formulated as sequence labeling: the correction system receives a string of input symbols and associates each input symbol with a (possibly empty) sequence of output symbols as shown in Figure 1. The input to the correction system can represent a line of text or an isolated word. We will only consider the case of isolated word correction in the present work.

This paper presents two models for supervised spelling correction. Both treat the task as sequence labeling but one of the models is structured and the other one is unstructured. Both systems are implemented as finite-state machines and are trained on data consisting of word pairs aligned at character level.

Our unstructured model is a finite-state transducer compiled from a set of weighted context-sensitive replace rules that are used to generate

Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata, pages 51–59,
Berlin, Germany, August 12, 2016.

correction candidates from input strings. These substitutions and their contexts are extracted from training data. This approach was first presented by Lindén (2006) for generating multilingual spelling variants of scientific and medical terms originating from Latin and Greek, but it also suitable for other tasks involving probabilistic string-to-string translation.

Our structured model is an averaged perceptron tagger. We represent the classifier as a composition of two weighted finite-state machines which incorporate the unstructured and structured features and parameters of the tagger. When these are combined with an input string, the resulting finite-state machine encodes all correction candidates with their respective weights assigned by the tagger. The finite-state implementation allows us to extract a given amount of the best scoring correction candidates using well-known and efficient algorithms that are widely available. The finite-state implementation also allows for restricting candidates to those found in a dictionary.

The paper is structured in the following way. Section 2 presents earlier approaches to spelling correction and the more general task of string-to-string translation. In Section 3, we present the unstructured and structured models used for spelling correction. In Section 4, we present the features utilized by the correction systems and in Section 5, we show how the systems can be implemented using finite-state methods. In Section 6, we present the data sets used in the experiments and in Section 7, we present the experimental setup of the paper and the results of the experiments. Finally, we discuss the results in Section 8 and conclude the paper in Section 9.

2 Related Work

Spelling correction is an old NLP task. The earliest approaches used plain edit distance combined with a lexicon. The edit distance approach was refined by Brill and Moore (2000) who added weights for edit operations. These systems ignored the context of the edit operation, which can nevertheless be quite useful.

Dreyer et al. (2008) investigate string-to-string translation which is a more general task than spelling correction. In order to incorporate symbol contexts into their models, they formulate string-to-string translation as a sequence labeling task. Their sequence labeling model is discriminative and the alignment between the input and output string is a latent variable. Dreyer et al. (2008) implement their model as a finite-state machine. This model is similar to ours but we do not treat the alignment between input and output strings as a latent variable. Instead, the training data for our model is aligned in advance.

Another interesting approach is presented by Xu et al. (2014) who learn a number of weighted rewrite rules from data. They use a log linear model to combine the rules and treat the alignment of the input and output forms as a latent variable like Dreyer et al. (2008). This system is reminiscent of ours because we also implement our two systems using rewrite rules. However, again, we do not treat the alignment of the input and output strings as a hidden variable.

Hulden and Francom (2013) compare two FST based methods for Spanish-language tweet normalization. The first method relies on a hierarchically arranged set of unweighted context-sensitive replace rules, while their other approach utilizes a noisy-channel FST model on the input string. These operations and their weights are extracted from the training data set. The authors report a somewhat better performance for the unweighted rule-based method, with a final accuracy of 60 %, but note that there is no theoretical obstacle that would prevent the inclusion of contexts in the weighted model.

Han and Baldwin (2011) present a method for recognizing and correcting out-of-vocabulary words in tweets and SMSs. They perform a set of normalization processes to the input word to make the relationship between the incorrect form and the correct form more transparent and generate a set of candidates within a certain edit distance. In addition to comparing orthographic forms, they also consider the phonetic realization of the input word and find correction candidates whose pronunciation is within a certain edit distance from the pronunciation of the input word.

In recent work, Eger et al. (2016) survey four systems for string-to-string translation on spelling correction of Tweets and normalization of historical Latin text. (1) The Sequitur system (Bisani and Ney, 2008) implements a joint generative model on input and output strings using *graphones*, which are units consisting of one input symbol and a possibly empty sequence of output symbols. (2) The DirecTL+ (Jiampojamarn et al.,

2010) represents the translation task as a pipeline of a string segmentation system, which splits the input string into character sequences, and a discriminative sequence labeling system which translates the character sequences into output symbols. DirecTL+ utilizes joint character n-grams in the discriminative sequence labeling system. (3) The AliSeTra system is based on the work of Eger (2012). Like DirecTL+, it also views string-to-string translation as a pipeline of segmentation and sequence labeling. (4) The final system surveyed by Eger et al. (2016) represents the string-to-string translation task as a series of contextual edit operations on the input string (Cotterell et al., 2015). The operations are compiled into a weighted finite-state machine. The edit operations are weighted using a probabilistic model which resembles the maximum entropy Markov model (MEMM) (McCallum et al., 2000). This system is similar to our structured system but we use a different feature set and estimate weights using the average perceptron algorithm. This avoids the well-known label bias problem (Lafferty et al., 2001) associated with MEMMs.

Systems 1, 2 and 3 surveyed by Eger et al. (2016) form an interesting contrast to our systems because we do not use segmentation of the input string. In this sense, our system is simpler.

Eger et al. (2016) present experiments on spelling correction both for the individual systems discussed in the paper and also various combinations of the systems. The AliSeTra system is shown to give the best performance of all individual surveyed systems on both Twitter data and historical Latin. We also present experiments on the Twitter data used by Eger et al. (2016) and show that our structured system delivers at least the same level of performance as the AliSeTra system.

3 Models

In the following, we present the unstructured and structured models used for spelling correction. Both models express the probability $p(y|x)$ of a normalization sequence $y = (y_1, ..., y_T)$ given an input sequence $x = (x_1, ..., x_T)$.

The input sequence x and output sequence y are formally required to have the same length. In practice, each element of y can, however, consist of a number of characters. This allows modeling of insertions and deletions. For example in Figure 1, the input sequence is $(\mathrm{A}, \mathrm{m}, ., \mathrm{c})$ and the output sequence is $(\mathrm{A}, \mathrm{nn}, \varepsilon, \mathrm{e})$. This corresponds to a deletion $. \rightarrow \varepsilon$ and a substitution $\mathrm{m} \rightarrow \mathrm{nn}$. The model cannot directly express the substitution of two consecutive input symbols with one output, for example $\mathrm{nn} \rightarrow \mathrm{m}$. This can, however, be expressed indirectly using a deletion and subsequent substitution as in $\mathrm{n} \rightarrow \varepsilon$ and $\mathrm{n} \rightarrow \mathrm{m}$.

The parameters of the models are estimated in a supervised manner using training data consisting of pairs of input and output strings. In order to accelerate training, we use aligned training data (consisting of symbol pairs) instead of treating the alignment of input and output strings as a latent variable.

3.1 Unstructured Classifier

This classifier represents the conditional probability $p(y|x)$ of a normalization $y = (y_1, ..., y_T)$ given an input $x = (x_1, ..., x_T)$ in an unstructured manner, that is

$$p(y|x) = \prod_{t=1}^{T} p(y_t|x, t)$$

This corresponds to making the assumption that output symbols y_t and y_u $(t \neq u)$ are independent given the input x.

To determine the probabilities $p(y_t, |x, t)$, we first map each input position (x, t) to a context $L \diamond x_t \diamond R$, where L and R are regular languages, and the input position (x, t) matches $L \diamond x_t \diamond R$, that is

$$x_1 \ ... \ x_{t-1} \diamond x_t \diamond x_{t+1} \ ... \ x_T \in L \diamond x_t \diamond R.$$

The $\diamond$ symbol is a special symbol which does not occur in any input string or output string.

We then define

$$p(y_t|x, t) = p(y_t|L \diamond x_t \diamond R)$$

where the probability $p(y_t|L \diamond x_t \diamond R)$ is estimated from the training data simply by counting occurrences of output symbols z in positions which match $L \diamond x_t \diamond R$. More precisely,

$$p(z|L \diamond x_t \diamond R) =$$

$$\frac{|\{(x, t) \text{ matches } L \diamond x_t \diamond R \text{ and } y_t = z\}|}{|\{(x, t) \text{ matches } L \diamond x_t \diamond R\}|}$$

Every input position encountered during test time should be mapped to a unique context. Therefore, the collection of contexts $L \diamond x_t \diamond R$ is chosen

in such a way that it forms a partition of $\Sigma^* \diamond \Sigma \diamond \Sigma^*$, where Σ is the set of all input symbols. In Section 4, we give a more detailed explanation of how these contexts are chosen.

3.2 Peceptron Tagger

Our structured spelling correction system is formulated as a traditional averaged perceptron tagger (Collins, 2002) as shown in Equation 1. Given an input sequence x of length T, the model assigns a score $\mathrm{s}(\cdot)$ for each output sequence y of length T as determined by the model parameters $\boldsymbol{w}$ and a vector valued feature extraction function $\boldsymbol{\phi}$. The n best normalization candidates given by the system can be extracted by finding the n highest scoring outputs y.

$$\mathrm{s}(x, y; \boldsymbol{w}) = \sum_{t=1}^{T} \boldsymbol{w} \cdot \boldsymbol{\phi}(y_{t-2}, y_{t-1}, y_t, x, t) \quad (1)$$

The labels y_{-1} and y_0 required for Equation 1 are word boundary symbols.

4 System Specification

This section presents the contexts used for the unstructured correction system and the features used by the structured correction system.

4.1 Contexts for the Unstructured Classifier

As explained in Section 3.1, the unstructured normalization model maps each input position (x, t) to a context $L \diamond x_t \diamond R$. These contexts form a partition of $\Sigma^* \diamond \Sigma \diamond \Sigma^*$.

The inventory of contexts is controlled by hyper-parameters which are determined using held-out data: n_{TH} which is a minimum number of context occurrences in the training data. Another parameter is l_C which is the length of the maximal right-hand context. We have set the value of l_C as 2 based on preliminary experiments.

If $x_{t-1}x_t x_{t+1}...x_{t+l_C}$ occurs at least n_{TH} times in the training data,

$$\Sigma^* x_{t-1} \diamond x_t \diamond x_{t+1}...x_{t+l_C} \Sigma^*$$

is chosen as context. If it occurs fewer times, each of the sub-strings $x_{t-1}x_t x_{t+1}...x_{t+k}$, where $0 \leq k < n_C$ is considered in turn. The longest one that occurs at least n_{TH} times in the training data is used to define a context. If none of them occur more than n_{TH} times, the single symbol x_t is used to define the context.

For each context $L \diamond x_t \diamond R$, we include a number of back-off contexts. For example, let $\Sigma^* a \diamond x_t \diamond b\ c\ \Sigma^*$ be a context, then back-off contexts are the following contexts.

$\Sigma^*\ a \diamond x_t \diamond b\ \Sigma^*$
$\Sigma^*\ a \diamond x_t \diamond \Sigma^*$
$\Sigma^* \diamond x_t \diamond \Sigma^*$

In order to ensure that no two contexts overlap, we need to modify the contexts slightly:

$\Sigma^*\ a \diamond x_t \diamond b\ [\Sigma - c]\ \Sigma^*$
$\Sigma^*\ a \diamond x_t \diamond [\Sigma - b]\ \Sigma^*$
$\Sigma^*\ [\Sigma - a]\ \diamond x_t \diamond \Sigma^*$

4.2 Features for the Perceptron Tagger

The structured correction system extracts unstructured and structured features from the input and output context of letters. Unstructured features associate the output in a single position with letters in the input. In contrast, structured features associate output letters with each other. Given an input string $x = (x_1, ..., x_T)$ and an output string $y = (y_1, ..., y_T)$, the unstructured features extracted at position t are

1. (x_t, y_t)
2. (x_{t-1}, x_t, y_t) and (x_t, x_{t+1}, y_t)
3. $(x_{t-3}, x_{t-2}, x_{t-1}, y_t)$, $(x_{t-2}, x_{t-1}, x_t, y_t)$, $(x_{t-1}, x_t, x_{t+1}, y_t)$, $(x_t, x_{t+1}, x_{t+1}, y_t)$ and $(x_{t+1}, x_{t+2}, x_{t+3}, y_t)$

In addition, we extract the structured features

1. (y_t)
2. (y_{t-1}, y_t)
3. (y_{t-2}, y_{t-1}, y_t)

The unstructured features are aimed at capturing the context of edit operations. Meanwhile, the structured features act as a language model.

5 Implementation

This section describes the finite-state implementation of our correction systems as weighted replace rules (Mohri and Sproat, 1996). Formally, the systems can be seen as sets of weighted *parallel* replace rules. As explained below, we however implement them using a *cascade* of weighted rules for efficiency reasons. This section will also describe the combination of replace rules and lexicon which is used in some of the experiments.

5.1 Weighted Parallel Replace Rules

Consider the following rule in XFST syntax (Beesley and Karttunen, 2003)

$$\mathrm{u} \rightarrow \varepsilon ::0.05 \;\; || \;\; \mathrm{u} \; \_$$

The rule matches in a context where the input contains two consecutive symbols u, deletes the second of them and assigns a penalty weight of $0.05 \approx -\log(0.95)$. The HFST library (Lindén et al., 2011) implements these weighted rules.

The unstructured system described in Section 3.1 uses a set of mutually exclusive features as explained in Section 4.1. Conceptually, the system can therefore be seen as a set of parallel replace rules (Kempe and Karttunen, 1996) acting on the same input strings. Although this formulation is theoretically pleasing and weighted parallel replace rules are available through the HFST interface (Lindén et al., 2011), preliminary experiments revealed that compilation of the system represented using parallel replace rules is slow in presence of training data of realistic scope. However, the subset of parallel replace rules needed in our two systems can be reformulated as normal replace rules to take advantage of a sequence of compose operations eliminating the speed issue in practice, see Section 5.3.

5.2 Unstructured Rules

The formulation of the substitutions and the contexts as explained in 4.1 as parallel replace rules is fairly straightforward. For instance, the substitution x_t with z in the context $\Sigma^* a \diamond x_t \diamond \; b \; c \; \Sigma^*$ is accomplished by the rule

$$x_t \rightarrow z ::w \;\; \| \; a \; \_ \; b \; c$$

Rules are assigned log weights which correspond to the probabilities p of the substitutions they express, i.e. $w = -log(p)$.

As explained in 5.1, rules are formulated as mutually exclusive by supplementing the contexts of the backoff rules with a negative expression containing the non-overlapping parts from the higher-order rule. This expression effectively blocks the lower-order rule if a higher-order rule can be applied instead. For instance, if the rule set contains the higher-order rule

$$x \rightarrow z ::0.4 \;\; \| \; x \; \_ \; y \; z$$

and a backoff rule

$$a \rightarrow b ::0.2 \;\; \| \; x \; \_ \; y$$

the latter is rewritten as

$$a \rightarrow b ::0.2 \;\; \| \; x \; \_ \; y \; [\; ? \; - \; z \;]$$

Note that deletions and insertions are treated here as ordinary substitutions, and the empty string ε is thus treated like any other symbol. The weights for insertions such as $\varepsilon \rightarrow \mathrm{a}$ and non-insertions ($\varepsilon \rightarrow \varepsilon$) are estimated accordingly. The sole exception to this are context-free insertions that, unlike other context-free substitutions, are disallowed altoghether.

5.3 Cascaded Weighted Rewrite Rules

In order to avoid the slow compilation of general parallel replace rules, we can reformulate the problem using a cascade of replace rules. In order to maintain the correct semantics of the system in a cascaded setting, we formulate the input and output in such a way that rules no longer perform translation of the input string. Instead the input already encodes all possible outputs and rules simply assign weights to alternative output candidates. In practice, we represent inputs as sequences of pairs separated[1] by a special symbol $\bullet$ which is neither an input nor a potential output symbol. Let us look at the following regular expression in Xerox syntax:

$$\bullet \; \# \; \# \; \bullet \; \mathrm{t} \; [\mathrm{t}|\mathrm{th}|\mathbb{O}] \; \bullet \; \mathrm{e} \; [\mathrm{e}|\mathrm{c}|\mathbb{O}] \; \bullet \; \# \; \# \; \bullet$$

The $\bullet$ symbol unambiguously outlines the sequence of input and output symbol pairs. The first pair of the sequence contains the word boundary symbols $\#$. Before feature extraction, we pad the aligned strings with this auxiliary symbol in order to formulate correspondences occurring in string-inital and and string-final positions.The second pair of the sequence contains an input symbol t and a set of potential output symbols, of which the $\mathbb{O}$ symbol denotes a deletion.

Using this representation, rules can be reformulated as weighting expressions. For example, the rule

$$\mathbb{O} \rightarrow \mathbb{O} ::0.05 \;\; || \;\; \mathrm{t} \; ? \; \bullet \; e \; \_ \; \bullet$$

assigns a penalty of 0.05 to a deletion of the second of the input symbols in our example above.

Features are composed into a weighted transducer W. Given an input I in the format presented above, an n-best algorithm (Allauzen et al., 2007) can extract the best scoring paths of $I \circ W$, from which the output strings are extracted.

[1] The inventory of pairs is extracted from the training data

5.4 Structured Rules

The structured classifier uses both unstructured and structured features. As seen above, unstructured features can be compiled into replace rules. Structured features can also be formulated as rules. For example, the following rule assigns a penalty to the output sequence "thh":

$$\mathrm{h} \rightarrow \mathrm{h}:: -33126 \quad || \quad ?\ \mathrm{t}\ \bullet\ ?\ \mathrm{h}\ \bullet\ ?\ \_\ \bullet$$

Both the unstructured and the structured systems apply rules in the same way. Unstructured and structured features are composed respectively giving us two weighted transducers U and S. Given an input I in the format presented above, we extract the best scoring paths from $I \circ U \circ S$.

5.5 Minimization of Transducers with Weights in the Tropical Semiring

We use finite-state machines with weights in the tropical weight semiring as defined by Allauzen et al. (2007). Because we use a series of compositions spanning several thousands of rule transducers for compiling the unstructured and structured feature transducers U and S, efficient determinization and minimization algorithms are crucial.

The minimization algorithm presented by Mohri and Sproat (1996) is available through the HFST interface and applicable to transducers with tropical weights where the weights are non-negative. Unfortunately, the structured correction system incorporates both positive and negative weights.

One solution to this problem is provided by Eisner (2003) who introduces a more general formulation of transducer minimization which is applicable to transducers with tropical weights in the entire range $\mathbb{R} \cup \{\infty, -\infty\}$ and many other weight classes as well. We have, however, resorted to a simpler approach which is applicable in the special case of tropical weights. After epsilon removal and determinization but before minimization, we traverse the transitions and the final state of the transducer M once and find the minimal weight w_{min}. Subsequently, we increment all transition and final weights in transducer M by $|w_{min}|$ which results in a transducer M^+ with non-negative weights.

Let $w_{M(p)}$ be the weight assigned by transducer M to path p. It is easy to see that $w_{M^+(p)} = w_{M(p)} + |p| \cdot |w_{min}|$, where $|p|$ is the length of p. We then apply conventional minimization resulting in a machine N^+. Subsequently, we subtract the weight $|w_{min}|$ from each transition in N^+ resulting in a machine N. As long as M does not contain any epsilon transitions, the length of each path p must be preserved by minimization. Therefore the total weight of each path p

$$\begin{aligned} \mathrm{w}_N(p) &= \mathrm{w}_M(p) + |p| \cdot |w_{min}| - |p| \cdot |w_{min}| \\ &= \mathrm{w}_M(p) \end{aligned}$$

is also preserved. Consequently, the minimized machine accepts the same weighted relation as the original machine.

5.6 Using a Lexicon

Some of our experiments utilize a lexicon. Preliminary experiments indicated that the lexicon should be combined in different ways with the unstructured and structured system.

When using a lexicon, the unstructured system returns the highest scoring correction candidate which is found in the lexicon. If none of the candidates are found in the lexicon, the system returns the input form. In the unstructured system, the task of an ouput language model is carried by the lexicon alone.

The structured system extracts the N highest scoring correction candidates and returns the highest scoring one of these that is found in the lexicon. If none of the candidates are found in the lexicon, the system returns the highest scoring candidate, which is plausible when part of the ouput language model is encoded by the structured features assuming that the lexicon is incomplete. This setup was also used by Eger et al. (2016).

6 Data and Resources

We perform experiments on two data sets: a collection of twitter spelling errors[2] used by Eger et al. (2016) and a corpus of Early Modern Finnish scanned texts that have been processed using an OCR engine. The data sets differ in the sense that the Twitter data contains only spelling errors but the Early Modern Finnish corpus contains a large number of correctly recognized forms in addition to OCR errors. Following Eger et al. (2016), we use only the first 5000 word pairs from the Twitter data set.

Both data sets consist of word pairs where the first word is the original word and the second one

[2] Available from `http://luululu.com/tweet/`.

is its normalization. As our systems are trained on aligned data, we used the grapheme to phoneme translation system Phonetisaurus (Novak et al., 2016) to align input and output strings used for training.

The Finnish data used in our experiments constitutes a part of a larger corpus of historical newspapers and magazines (KLK) that has been digitized by the National Library of Finland. Some of this digitized material has been manually corrected and edited at the Institute for the Languages of Finland. Further correction has been carried out via crowd-sourcing. Our data set consists of running text extracted from the OCR processed 19th-century publications for which manually edited material is available and comprises roughly 40 000 OCR processed word pairs.

We perform tenfold cross-validation on both data sets. We divide the data sets into ten non-overlapping parts $D_1, ..., D_{10}$ in the following way. For each consecutive ten word pairs (starting with the first), we assign the pair at position i to set D_i. We then form ten training, development and test sets. The test set E_i is D_i. The development set U_i is T_{i-1} when $i > 1$ and T_{10} when $i = 1$. The training set T_i consists of the remaining partitions D_j. Hence training set T_i, development set U_i and test set E_i never overlap.

A lexicon is required for some of the experiments. For Twitter data, we use the lexicon ColLex.EN (Brück et al., 2014) following Eger et al. (2016), and for the Finnish OCR data, we use the OMorFi open-source Finnish morphological analyzer (Pirinen, 2008).

For the task of correcting text written in Early Modern Finnish, the OMorFi analyzer had to be modified slightly to recognize capitalized variants of all word forms as well as to accept some of the more archaic spelling variants and vocabulary found in Early Modern Finnish. We did this by supplementing the acceptor with word forms extracted from the KLK corpus whose frequency equals or exceeds 100. Word forms already accepted by OMorFi were given precedence.

7 Experiments

We perform experiments on the Twitter data and Early Modern Finnish OCR data in the same manner. For each data set, we measure the performance of the unstructured and structured correction system using tenfold cross-validation on the data splits presented in Section 6. Our data sets and code are available online.[3]

tp	Number of erroneous inputs which are corrected.
fp	Number of correct inputs which are changed to an incorrect output.
fn	Number of erroneous input which are not corrected.

Table 1: Definition of edit types.

Both of the models presented in the paper incorporate hyper-parameters. We first set the hyper-parameters using development data, then combined the development and training data and use the combination to train the final system which is used to process the test data.

The FinnPos tagger toolkit (Silfverberg et al., 2015) is used to train the models for the structured system and the HFST Python interface (Lindén et al., 2011) is used for constructing and operating finite-state machines. When training FinnPos models, we used default settings for most hyper-parameters. Only the number of training epochs is determined using development data. For experiments using a lexicon, we additionally use development data to set the number of top correction candidates N which are looked up in the lexicon as explained in Section 5.6. For tweet normalization, we use $N = 80$ and for Finnish OCR post processing, we use $N = 5$.

As an evaluation metrics, we use *correction rate* (CR) defined as

$$\mathrm{CR} = \frac{tp - fp}{tp + fn}$$

where tp, fp and fn are defined in Table 1. Note that when all input forms are incorrect (as in the case of the Twitter data), CR corresponds exactly to the evaluation metric word accuracy (WACC) used by Eger et al. (2016) because the count fp is 0.

$$\mathrm{WACC} = \frac{tp}{tp + fn}$$

7.1 Results

Tables 2 and 3 show the results of the experiments on the Finnish OCR data and Twitter data. We perform tenfold cross-validation and provide t-based confidence intervals at the 95% level.

[3] https://github.com/mpsilfve/ocrpp

	No lexicon (%)	Lexicon (%)
UC	48.56 ± 2.00	57.80 ± 1.82
AliSeTra	68.38 ± 1.52	72.98 ± 2.01
PT	$\mathbf{70.14 \pm 1.43}$	$\mathbf{74.66 \pm 1.38}$

Table 2: Results for tweet normalization. UC refers to the unstructured classifier presented in Section 3.1, PT to the perceptron tagger presented in Section 3.2 and AliSeTra to the system presented by Eger et al. (2016).

	No lexicon (%)	Lexicon (%)
UC	20.02 ± 1.29	21.58 ± 2.11
PT	$\mathbf{32.05 \pm 1.97}$	$\mathbf{35.09 \pm 2.08}$

Table 3: Results for Finnish historical OCR.

For the Finnish OCR data, the structured perceptron correction system clearly outperforms the unstructured system both without using a lexicon and when using a lexicon. The difference in performance is statistically significant in both cases at the 95% confidence level. Because the AliSeTra system is not freely available, we do not have results for that system on the Finnish OCR data.

For the Twitter data, both AliSeTra and the perceptron tagger deliver superior accuracy when compared with the unstructured system. The average performance of the perceptron tagger in this experiment is superior to the performance of the AliSeTra system as reported by Eger et al. (2016). The difference in performance is, however, not statistically significant. It should be noted that the data splits used in this work differ from the splits used by Eger et al. (2016).

8 Discussion

The advantage of the unstructured approach is that relatively little time is required for training the error model (on the league of ten minutes for our data sets). The drawback of the unstructured models used is that they accommodate a very limited set of features which manifests as comparably low normalization performance.

The performance of our structured system is at least equal to recent work of Eger et al. (2016) on tweet normalization even though our system is simpler in the sense that we use a 1-to-n mapping from inputs to outputs whereas Eger et al. (2016) use an n-to-n alignment between the input and output. The n-to-n alignment, however, requires segmentation of the input as a preprocessing step. This may induce errors which cannot be corrected later. Treating the segmentation as a latent variable following (Cotterell et al., 2015) could be a solution but it carries the disadvantage of slow estimation and inference.

It should be noted that we use some additional unstructured features compared with Eger et al. (2016) which may explain our slight performance advantage.

9 Conclusions

We have presented two systems for word based spelling correction using finite-state methods. We have shown that we reach state-of-the-art results when compared with a recent system presented by Eger et al. (2016) on the task of tweet normalization. Additionally, we have presented experiments on the task of OCR post-processing on a corpus consisting of Early Modern Finnish newspaper text.

10 Acknowledgments

We wish to thank the anonymous reviewers for their valuable suggestions.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, CIAA'07, pages 11–23, Berlin, Heidelberg. Springer-Verlag.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*, volume 3 of *CSLI Studies in Computational Linguistics*. CSLI Publications.

Maximilian Bisani and Hermann Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Commun.*, 50(5):434–451, May.

Eric Brill and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 286–293, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tim Vor Der Brück, Alexander Mehler, and Zahurul Islam. 2014. Collex.en: Automatically generating and evaluating a full-form lexicon for english. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente

Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, may. European Language Resources Association (ELRA).

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *Transactions of the Association for Computational Linguistics*, 3:433–447.

Markus Dreyer, Jason R. Smith, and Jason Eisner. 2008. Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1080–1089, Stroudsburg, PA, USA. Association for Computational Linguistics.

Steffen Eger, Tim vor der Brck, and Alexander Mehler. 2016. A comparison of four character-level string-to-string translation models for (ocr) spelling error correction. *The Prague Bulletin of Mathematical Linguistics*, 105:77–99.

Steffen Eger. 2012. S-restricted monotone alignments: Algorithm, search space, and applications. In *Proceedings of COLING 2012*, pages 781–798, Mumbai, India, December. The COLING 2012 Organizing Committee.

Jason Eisner. 2003. Simpler and more general minimization for weighted finite-state automata. In *HLT-NAACL*.

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: Makn sens a# twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 368–378. Association for Computational Linguistics.

Mans Hulden and Jerid Francom. 2013. Weighted and unweighted transducers for tweet normalization. In *Tweet-Norm@ SEPLN*, pages 69–72. Citeseer.

Sittichai Jiampojamarn, Colin Cherry, and Grzegorz Kondrak. 2010. Integrating joint n-gram features into a discriminative training framework. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, pages 697–700.

André Kempe and Lauri Karttunen. 1996. Parallel replacement in finite state calculus. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96, pages 622–627, Stroudsburg, PA, USA. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A Pirinen, and Miikka Silfverberg. 2011. HFSTFramework for Compiling and Applying Morphologies. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology*, volume 100 of *Communications in Computer and Information Science*, pages 67–85. Springer Berlin Heidelberg.

Krister Lindén. 2006. Multilingual modeling of cross-lingual spelling variants. *Inf. Retr.*, 9(3):295–310, June.

Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. 2000. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 591–598, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mehryar Mohri and Richard Sproat. 1996. An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 231–238, Santa Cruz, California, USA, June. Association for Computational Linguistics.

Josef Robert Novak, Nobuaki Minematsu, and Keikichi Hirose. 2016. Phonetisaurus: Exploring grapheme-to-phoneme conversion with joint n-gram models in the wfst framework. *Natural Language Engineering*, FirstView:1–32, 4.

Tommi Pirinen. 2008. Automatic finite state morphological analysis of Finnish language using open source resources (in Finnish). Master's thesis, University of Helsinki.

Miikka Silfverberg, Teemu Ruokolainen, Krister Lindn, and Mikko Kurimo. 2015. Finnpos: an open-source morphological tagging and lemmatization toolkit for finnish. *Language Resources and Evaluation*, pages 1–16.

Gu Xu, Hang Li, Ming Zhang, and Ziqi Wang. 2014. A probabilistic approach to string transformation. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1–1.