A Discriminative Candidate Generator for String Transformations

Naoaki Okazaki[†]
okazaki@is.s.u-tokyo.ac.jp

Yoshimasa Tsuruoka[‡]
yoshimasa.tsuruoka@manchester.ac.uk

Sophia Ananiadou[‡]
sophia.ananiadou@manchester.ac.uk

Jun'ichi Tsujii[†‡]
tsujii@is.s.u-tokyo.ac.jp

[†]Graduate School of Information Science and Technology
University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo 113-8656, Japan

[‡]School of Computer Science,
University of Manchester
National Centre for Text Mining (NaCTeM)
Manchester Interdisciplinary Biocentre
131 Princess Street, Manchester M1 7DN, UK

Abstract

String transformation, which maps a source string s into its desirable form t^*, is related to various applications including stemming, lemmatization, and spelling correction. The essential and important step for string transformation is to generate candidates to which the given string s is likely to be transformed. This paper presents a discriminative approach for generating candidate strings. We use substring substitution rules as features and score them using an L_1-regularized logistic regression model. We also propose a procedure to generate negative instances that affect the decision boundary of the model. The advantage of this approach is that candidate strings can be enumerated by an efficient algorithm because the processes of string transformation are tractable in the model. We demonstrate the remarkable performance of the proposed method in normalizing inflected words and spelling variations.

1 Introduction

String transformation maps a source string s into its destination string t^*. In the broad sense, string transformation can include labeling tasks such as part-of-speech tagging and shallow parsing (Brill, 1995). However, this study addresses string transformation in its narrow sense, in which a part of a source string is rewritten with a substring. Typical applications of this task include stemming, lemmatization, spelling correction (Brill and Moore, 2000; Wilbur et al., 2006; Carlson and Fette, 2007), OCR error correction (Kolak and Resnik, 2002), approximate string matching (Navarro, 2001), and duplicate record detection (Bilenko and Mooney, 2003).

Recent studies have formalized the task in the discriminative framework (Ahmad and Kondrak, 2005; Li et al., 2006; Chen et al., 2007),

$$t^* = \operatorname*{argmax}_{t \in \mathrm{gen}(s)} P(t|s). \quad (1)$$

Here, the *candidate generator* $\mathrm{gen}(s)$ enumerates candidates of destination (correct) strings, and the *scorer* $P(t|s)$ denotes the conditional probability of the string t for the given s. The scorer was modeled by a noisy-channel model (Shannon, 1948; Brill and Moore, 2000; Ahmad and Kondrak, 2005) and maximum entropy framework (Berger et al., 1996; Li et al., 2006; Chen et al., 2007).

The candidate generator $\mathrm{gen}(s)$ also affects the accuracy of the string transformation. Previous studies of spelling correction mostly defined $\mathrm{gen}(s)$,

$$\mathrm{gen}(s) = \{t \mid \mathrm{dist}(s,t) < \delta\}. \quad (2)$$

Here, the function $\mathrm{dist}(s,t)$ denotes the weighted Levenshtein distance (Levenshtein, 1966) between strings s and t. Furthermore, the threshold δ requires the distance between the source string s and a candidate string t to be less than δ.

The choice of $\mathrm{dist}(s,t)$ and δ involves a tradeoff between the precision, recall, and training/tagging speed of the scorer. A less restrictive design of these factors broadens the search space, but it also increases the number of confusing candidates, amount of feature space, and computational cost for the scorer. Moreover, the choice is highly dependent on the target task. It might be sufficient for a spelling

correction program to gather candidates from known words, but a stemmer must handle unseen words appropriately. The number of candidates can be huge when we consider transformations from and to unseen strings.

This paper addresses these challenges by exploring the discriminative training of candidate generators. More specifically, we build a binary classifier that, when given a source string s, decides whether a candidate t should be included in the candidate set or not. This approach appears straightforward, but it must resolve two practical issues. First, the task of the classifier is not only to make a binary decision for the two strings s and t, but also to enumerate a set of positive strings for the string s,

$$\mathrm{gen}(s) = \{t \mid \mathrm{predict}(s, t) = 1\}. \quad (3)$$

In other words, an efficient algorithm is necessary to find a set of strings with which the classifier $\mathrm{predict}(s, t)$ yields positive labels for the string s.

Another issue arises when we prepare a training set. A discriminative model requires a training set in which each instance (pair of strings) is annotated with a positive or negative label. Even though some existing resources (e.g., inflection table and query log) are available for positive instances, such resources rarely contain negative instances. Therefore, we must generate negative instances that are effective for discriminative training.

To address the first issue, we design features that express transformations from a source string s to its destination string t. Feature selection and weighting are performed using an L_1-regularized logistic regression model, which can find a sparse solution to the classification model. We also present an algorithm that utilizes the feature weights to enumerate candidates of destination strings efficiently. We deal with the second issue by generating negative instances from unlabeled instances. We describe a procedure to choose negative instances that affect the decision boundary of the classifier.

This paper is organized as follows. Section 2 formalizes the task of the candidate generator as a binary classification modeled by logistic regression. Features for the classifier are designed using the rules of substring substitution. Therefore, we can obtain, efficiently, candidates of destination strings and negative instances for training. Section 3 reports the remarkable performance of the proposed method in various applications including lemmatization, spelling normalization, and noun derivation. We briefly review previous work in Section 4, and conclude this paper in Section 5.

2 Candidate generator

2.1 Candidate classification model

In this section, we first introduce a binary classifier that yields a label $y \in \{0, 1\}$ indicating whether a candidate t should be included in the candidate set (1) or not (0), given a source string s. We express the conditional probability $P(y|s, t)$ using a logistic regression model,

$$P(1|s, t) = \frac{1}{1 + \exp\left(-\boldsymbol{\Lambda}^T \boldsymbol{F}(s, t)\right)}, \quad (4)$$

$$P(0|s, t) = 1 - P(1|s, t). \quad (5)$$

In these equations, $\boldsymbol{F} = \{f_1, ..., f_K\}$ denotes a vector of the Boolean feature functions; K is the number of feature functions; and $\boldsymbol{\Lambda} = \{\lambda_1, ..., \lambda_K\}$ presents a weight vector of the feature functions.

We obtain the following decision rule to choose the most probable label y^* for a given pair $\langle s, t \rangle$,

$$y^* = \underset{y \in \{0,1\}}{\mathrm{argmax}}\, P(y|s, t) = \begin{cases} 1 & \left(\boldsymbol{\Lambda}^T \boldsymbol{F}(s, t) > 0\right) \\ 0 & (\text{otherwise}) \end{cases}. \quad (6)$$

Finally, given a source string s, the generator function $\mathrm{gen}(s)$ is defined to collect all strings to which the classifier assigns positive labels:

$$\begin{aligned} \mathrm{gen}(s) &= \{t \mid P(1|s, t) > P(0|s, t)\} \\ &= \{t \mid \boldsymbol{\Lambda}^T \boldsymbol{F}(s, t) > 0\}. \end{aligned} \quad (7)$$

2.2 Substitution rules as features

The binary classifier can include any arbitrary feature. This is exemplified by the Levenshtein distance and distributional similarity (Lee, 1999) between two strings s and t. These features can improve the classification accuracy, but it is unrealistic to compute these features for every possible string, as in equation 7. For that reason, we specifically examine *substitution rules*, with which the process

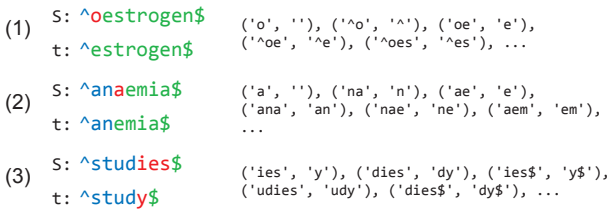

Figure 1: Generating substitution rules.

of transforming a source string s into its destination form t is tractable.

In this study, we assume that every string has a prefix '^' and postfix '$', which indicate the head and tail of a string. A substitution rule $r = (\alpha, \beta)$ replaces every occurrence of the substring α in a source string into the substring β. Assuming that a string s can be transformed into another string t with a single substitution operation, substitution rules express the different portion between strings s and t.

Equation 8 defines a binary feature function with a substitution rule between two strings s and t,

$$f_k(s,t) = \begin{cases} 1 & (\text{rule } r_k \text{ can convert } s \text{ into } t) \\ 0 & (\text{otherwise}) \end{cases}. \tag{8}$$

We allow multiple substitution rules for a given pair of strings. For instance, substitution rules ('a', ''), ('na', 'n'), ('ae', 'e'), ('nae', 'ne'), etc. form feature functions that yield 1 for strings $s =$ '^anaemia$' and $t =$ '^anemia$'. Equation 6 produces a decision based on the sum of feature weights, or scores of substitution rules, representing the different portions between s and t.

Substitution rules for the given two strings s and t are obtained as follows. Let l denote the longest common prefix between strings s and t, and r the longest common postfix. We define c_s as the substring in s that is not covered by the longest common prefix l and postfix r, and define c_t for t analogously. In other words, strings s and t are divided into three regions, lc_sr and lc_tr, respectively. For strings $s =$ '^anaemia$' and $t =$ '^anemia$' in Figure 1 (2), we obtain $c_s =$ 'a' and $c_t =$ '' because $l =$ '^an' and $r =$ 'emia$'.

Because substrings c_s and c_t express different portions between strings s and t, we obtain the *minimum substitution rule* (c_s, c_t), which can convert the string s into t by replacing substrings c_s in s with c_t; the minimum substitution rule for the same example is ('a', ''). However, replacing letters 'a' in '^anaemia$' into empty letters does not produce the correct string '^anemia$' but '^nemi$'. Furthermore, the rule might be inappropriate for expressing string transformation because it always removes the letter 'a' from every string.

Therefore, we also obtain *expanded substitution rules*, which insert postfixes of l to the head of minimum substitution rules, and/or append prefixes of r to the rules. For example, we find an expanded substitution rule ('na', 'n'), by inserting a postfix of $l =$ '^an' to the head of the minimum substitution rule ('a', ''); similarly, we obtain an expanded substitution rule ('ae', 'e'), by appending a prefix of $r =$ 'emia$' to the tail of the rule ('a', '').

Figure 1 displays examples of substitution rules (the right side) for three pairs of strings (the left side). Letters in blue, green, and red respectively represent the longest common prefixes, longest common postfixes, and different portions. In this study, we expand substitution rules such that the number of letters in rules is does not pass a threshold θ[1].

2.3 Parameter estimation

Given a training set that consists of N instances, $\mathcal{D} = \left((s^{(1)}, t^{(1)}, y^{(1)}), ..., (s^{(N)}, t^{(N)}, y^{(N)})\right)$, we optimize the feature weights in the logistic regression model by maximizing the log-likelihood of the conditional probability distribution,

$$\mathcal{L}_{\mathbf{\Lambda}} = \sum_{i=1}^{N} \log P(y^{(i)} | s^{(i)}, t^{(i)}). \tag{9}$$

The partial derivative of the log-likelihood with respect to a feature weight λ_k is given as equation 10,

$$\frac{\partial \mathcal{L}_{\mathbf{\Lambda}}}{\partial \lambda_k} = \sum_{i=1}^{N} \left\{ y^{(i)} - P(1 | s^{(i)}, t^{(i)}) \right\} f_k(s^{(i)}, t^{(i)}). \tag{10}$$

The maximum likelihood estimation (MLE) is known to suffer from overfitting the training set. The

[1]The number of letters for a substitution rule $r = (\alpha, \beta)$ is defined as the sum of the quantities of letters in α and β, i.e., $|\alpha| + |\beta|$. We determined the threshold $\theta = 12$ experimentally.

common approach for addressing this issue is to use the maximum a posteriori (MAP) estimation, introducing a regularization term of the feature weights $\mathbf{\Lambda}$, i.e., a penalty on large feature weights. In addition, the generation algorithm of substitution rules might produce inappropriate rules that transform a string incorrectly, or overly specific rules that are used scarcely. Removing unnecessary substitution rules not only speeds up the classifier but also the algorithm for candidate generation, as presented in Section 2.4.

In recent years, L_1 regularization has received increasing attention because it produces a sparse solution of feature weights in which numerous feature weights are zero (Tibshirani, 1996; Ng, 2004). Therefore, we regularize the log-likelihood with the L_1 norm of the weight vector $\mathbf{\Lambda}$ and define the final form the objective function to be minimized as

$$E_{\mathbf{\Lambda}} = -\mathcal{L}_{\mathbf{\Lambda}} + \frac{|\mathbf{\Lambda}|}{\sigma}. \quad (11)$$

Here, σ is a parameter to control the effect of L_1 regularization; the smaller the value we set to σ, the more features the MAP estimation assigns zero weights to: it removes a number of features from the model. Equation 11 is minimized using the Orthant-Wise Limited-memory Quasi-Newton (OW-LQN) method (Andrew and Gao, 2007) because the second term of equation 11 is not differentiable at $\lambda_k = 0$.

2.4 Candidate generation

The advantage of our feature design is that we can enumerate strings to which the classifier is likely to assign positive labels. We start by observing the necessary condition for t in equation 7,

$$\mathbf{\Lambda}^T \boldsymbol{F}(s,t) > 0 \Rightarrow \exists k : f_k(s,t) = 1 \wedge \lambda_k > 0. \quad (12)$$

The classifier might assign a positive label to strings s and t when at least one feature function whose weight is positive can transform s to t.

Let R^+ be a set of substitution rules to which MAP estimation has assigned positive feature weights. Because each feature corresponds to a substitution rule, we can obtain $\mathrm{gen}(s)$ for a given string s by application of every substitution rule $r \in R^+$,

$$\mathrm{gen}(s) = \{r(s) \mid r \in R^+ \wedge \mathbf{\Lambda}^T \boldsymbol{F}(s, r(s)) > 0\}. \quad (13)$$

Input: $s = (s_1, ..., s_l)$: an input string s (series of letters)
Input: D: a trie dictionary containing positive features
Output: T: $\mathrm{gen}(s)$

```
1  T = {};
2  U = {};
3  foreach i ∈ (1, ..., |s|) do
4      F ← D.prefix_search(s, i);
5      foreach f ∈ F do
6          if f ∉ U then
7              t ← f.apply(s);
8              if classify(s, t) = 1 then
9                  add t to T;
10             end
11             add f to U;
12         end
13     end
14 end
15 return T;
```

Algorithm 1: A pseudo-code for $\mathrm{gen}(s)$.

Here, $r(s)$ presents the string to which the substitution rule r transforms the source string s. We can compute $\mathrm{gen}(s)$ with a small computational cost if the MAP estimation with L_1 regularization reduces the number of active features.

Algorithm 1 represents a pseudo-code for obtaining $\mathrm{gen}(s)$. To search for positive substitution rules efficiently, the code stores a set of rules in a trie structure. In line 4, the code obtains a set of positive substitution rules F that can rewrite substrings starting at offset #i in the source string s. For each rule $f \in F$, we obtain a candidate string t by application of the substitution rule f to the source string s (line 7). The candidate string t is qualified to be included in $\mathrm{gen}(s)$ when the classifier assigns a positive label to strings s and t (lines 8 and 9). Lines 6 and 11 prevent the algorithm from repeating evaluation of the same substitution rule.

2.5 Generating negative instances

The parameter estimation requires a training set $\mathcal{D}$ in which each instance (pair of strings) is annotated with a positive or negative label. Negative instances (counter examples) are essential for penalizing inappropriate substitution rules, e.g. (‘a’, ‘’). Even though some existing resources (e.g. verb inflection table) are available for positive instances, such resources rarely contain negative instances.

A common approach for handling this situation is to assume that every pair of strings in a resource

Input: $\mathcal{D}^+ = [(s_1, t_1), ..., (s_l, t_l)]$: positive instances
Input: V: a suffix array of all strings (vocabulary)
Output: $\mathcal{D}^-$: negative instances
Output: R: substitution rules (features)

```
1  D⁻ = [];
2  R = {};
3  foreach d ∈ D⁺ do
4      foreach r ∈ features(d) do
5          add r to R;
6      end
7  end
8  foreach r ∈ R do
9      S ← V.search(r.src);
10     foreach s ∈ S do
11         t ← r.apply(s);
12         if (s, t) ∉ D⁺ then
13             if t ∈ V then
14                 append (s, t) to D⁻;
15             end
16         end
17     end
18 end
19 return D⁻, R;
```

Algorithm 2: Generating negative instances.

is a negative instance; however, negative instances amount to ca. $V(V-1)/2$, where V represents the total number of strings. Moreover, substitution rules expressing negative instances are innumerable and sparse because the different portions are peculiar to individual negative instances. For instance, the minimum substitution rule for unrelated words *anaemia* and *around* is ('naemia', 'round'), but the rule cannot be too specific to generalize the conditions for other negative instances.

In this study, we generate negative instances so that they can penalize inappropriate rules and settle the decision boundary of the classifier. This strategy is summarized as follows. We consider every pair of strings as candidates for negative instances. We obtain substitution rules for the pair using the same algorithm as that described in Section 2.2 if a string pair is not included in the dictionary (i.e., not in positive instances). The pair is used as a negative instance only when any substitution rule generated from the pair also exists in the substitution rules generated from positive instances.

Algorithm 2 presents the pseudo-code that implements the strategy for generating negative instances efficiently. First, we presume that we have positive instances $\mathcal{D}^+ = [(s_1, t_1), ..., (s_l, t_l)]$ and unlabeled strings V. For example, positive instance $\mathcal{D}^+$ represent orthographic variants, and unlabeled strings V include all possible words (vocabulary). We insert the vocabulary into a suffix array, which is used to locate every occurrence of substrings in V.

The algorithm first generates substitution rules R only from positive instances $\mathcal{D}^+$ (lines 3 to 7). For each substitution rule $r \in R$, we enumerate known strings S that contain the source substring $r.src$ (line 9). We apply the substitution rule to each string $s \in S$ and obtain its destination string t (line 11). If the pair of strings $\langle s, t \rangle$ is not included in D^+ (line 12), and if the destination string t is known (line 13), the substitution rule r might associate incorrect strings s and t, which do not exist in D^+. Therefore, we insert the pair to the negative set D^- (line 14).

Table	Description	# Entries
LRSPL	Spelling variants	90,323
LRNOM	Nominalizations (derivations)	14,029
LRAGR	Agreement and inflection	910,854
LRWD	Word index (vocabulary)	850,236

Table 1: Excerpt of tables in the SPECIALIST Lexicon.

Data set	# +	# -	# Rules
Orthography	15,830	33,296	11,098
Derivation	12,988	85,928	5,688
Inflection	113,215	124,747	32,278

Table 2: Characteristics of datasets.

3 Evaluation

3.1 Experiments

We evaluated the candidate generator using three different tasks: normalization of orthographic variants, noun derivation, and lemmatization. The datasets for these tasks were obtained from the *UMLS SPECIALIST Lexicon*[2], a large lexicon that includes both commonly occurring English words and biomedical vocabulary. Table 1 displays the list of tables in the SPECIALIST Lexicon that were used in our experiments. We prepared three datasets, *Orthography*, *Derivation*, and *Inflection*.

The Orthography dataset includes spelling variants (e.g., *color* and *colour*) in the LRSPL table. We

[2]UMLS SPECIALIST Lexicon: http://specialist.nlm.nih.gov/

chose entries as positive instances in which spelling variants are caused by (case-insensitive) alphanumeric changes[3]. The Derivation dataset was built directly from the LRNOM table, which includes noun derivations such as *abandon* → *abandonment*. The LRAGR table includes base forms and their inflectional variants of nouns (singular and plural forms), verbs (infinitive, third singular, past, past participle forms, etc), and adjectives/adverbs (positive, comparative, and superlative forms). For the Inflection dataset, we extracted the entries in which inflectional forms differ from their base forms[4], e.g., *study* → *studies*.

For each dataset, we applied the algorithm described in Section 2.5 to generate substitution rules and negative instances. Table 2 shows the number of positive instances (# +), negative instances (# -), and substitution rules (# Rules). We evaluated the performance of the proposed method in two different goals of the tasks: classification (Section 3.2) and normalization (Section 3.3).

3.2 Experiment 1: Candidate classification

In this experiment, we measured the performance of the classification task in which pairs of strings were assigned with positive or negative labels. We trained and evaluated the proposed method by performing ten-fold cross validation on each dataset[5]. Eight baseline systems were prepared for comparison: Levenshtein distance (LD), normalized Levenshtein distance (NLD), Dice coefficient on letter bigrams (DICE) (Adamson and Boreham, 1974), Longest Common Substring Ratio (LCSR) (Melamed, 1999), Longest Common Prefix Ratio (PREFIX) (Kondrak, 2005), Porter's stemmer (Porter, 1980), Morpha (Minnen et al., 2001), and CST's lemmatiser (Dalianis and Jongejan, 2006)[6].

The five systems LD, NLD, DICE, LCSR, and PREFIX employ corresponding metrics of string distance or similarity. Each system assigns a positive label to a given pair of strings ⟨s, t⟩ if the distance/similarity of strings s and t is smaller/larger than the threshold δ (refer to equation 2 for distance metrics). The threshold of each system was chosen so that the system achieves the best F1 score.

The remaining three systems assign a positive label only if the system transforms the strings s and t into the identical string. For example, a pair of two words *studies* and *study* is classified as positive by Porter's stemmer, which yields the identical stem *studi* for these words. We trained CST's lemmatiser for each dataset to obtain flex patterns that are used for normalizing word inflections.

To examine the performance of the L_1-regularized logistic regression as a discriminative model, we also built two classifiers based on the Support Vector Machine (SVM). These SVM classifiers were implemented by the SVM^{perf}[7] on a linear kernel[8]. An SVM classifier employs the same feature set (substitution rules) as the proposed method so that we can directly compare the L_1-regularized logistic regression and the linear-kernel SVM. Another SVM classifier incorporates the five string metrics; this system can be considered as our reproduction of the discriminative string similarity proposed by Bergsma and Kondrak (2007).

Table 3 reports the precision (P), recall (R), and F1 score (F1) based on the number of correct decisions for positive instances. The proposed method outperformed the baseline systems, achieving 0.919, 0.888, and 0.984 of F1 scores, respectively. Porter's stemmer worked on the Inflection set, but not on the Orthography set, which is beyond the scope of the stemming algorithms. CST's lemmatizer suffered from low recall on the Inflection set because it removed suffixes of base forms, e.g., (*cloning*, *clone*) → (*clone*, *clo*). Morpha and CST's lemma-

[3]LRSPL table includes trivial spelling variants that can be handled using simple character/string operations. For example, the table contains spelling variants related to case sensitivity (e.g., *deg* and *Deg*) and symbols (e.g., *Feb* and *Feb.*).

[4]LRAGR table also provides agreement information even when word forms do not change. For example, the table contains an entry indicating that the first-singular present form of the verb *study* is *study*, which might be readily apparent to English speakers.

[5]We determined the regularization parameter $\sigma = 5$ experimentally. Refer to Figure 2 for the performance change.

[6]We used CST's lemmatiser version 2.13: `http://www.cst.dk/online/lemmatiser/uk/index.html`

[7]SVM for Multivariate Performance Measures (SVM^{perf}): `http://svmlight.joachims.org/svm_perf.html`

[8]We determined the parameter $C = 500$ experimentally; it controls the tradeoff between training error and margin.

System	Orthography			Derivation			Inflection		
	P	R	F1	P	R	F1	P	R	F1
Levenshtein distance ($\delta = 1$)	.319	.871	.467	.004	.006	.005	.484	.679	.565
Levenshtein distance	.323	.999	.488	.131	1.00	.232	.479	.988	.646
Normalized Levenshtein distance	.441	.847	.580	.133	.990	.235	.598	.770	.673
Dice coefficient (letter bigram)	.401	.918	.558	.137	.984	.240	.476	1.00	.645
LCSR	.322	1.00	.487	.156	.841	.263	.476	1.00	.645
PREFIX	.418	.927	.576	.140	.943	.244	.476	1.00	.645
Porter stemmer (Porter, 1980)	.084	.074	.079	.197	.846	.320	.926	.839	.881
Morpha (Minnen et al., 2001)	.009	.007	.008	.012	.022	.016	.979	.836	.902
CST's lemmatiser (Dalianis et al. 2006)	.119	.008	.016	.383	.682	.491	.821	.176	.290
Proposed method	.941	.898	.919	**.896**	.880	.888	**.985**	.986	**.984**
Substitution rules trained with SVM	.943	.890	.916	.894	**.886**	**.890**	.980	**.987**	.983
+ LD, NLD, DICE, LCSR, PREFIX	**.946**	**.906**	**.926**	.894	**.886**	**.890**	.980	**.987**	.983

Table 3: Performance of candidate classification

Rank	Src	Dst	Weight	Examples
1	uss	us	9.81	focussing
2	aev	ev	9.56	mediaeval
3	aen	en	9.53	ozaena
4	iae$	ae$	9.44	gadoviae
5	nni	ni	9.16	prorennin
6	nne	ne	8.84	connexus
7	our	or	8.54	colour
8	aea	ea	8.31	paean
9	aeu	eu	8.22	stomodaeum
10	ooll	ool	7.79	woollen

Table 4: Feature weights for the Orthography set

tizer were not designed for orthographic variants and noun derivations.

Levenshtein distance ($\delta = 1$) did not work for the Derivation set because noun derivations often append two or more letters (e.g., *happy* → *happiness*). No string similarity/distance metrics yielded satisfactory results. Some metrics obtained the best F1 scores with extreme thresholds only to classify every instance as positive. These results imply the difficulty of the string metrics for the tasks.

The L_1-regularized logistic regression was comparable to the SVM with linear kernel in this experiment. However, the presented model presents the advantage that it can reduce the number of active features (features with non-zero weights assigned); the L_1 regularization can remove 74%, 48%, and 82% of substitution rules in each dataset. The performance improvements by incorporating string metrics as features were very subtle (less than 0.7%). What is worse, the distance/similarity metrics do not specifically derive destination strings to which the classifier is likely to assign positive labels. Therefore, we can no longer use the efficient algorithm as a candidate generator (in Section 2.4) with these features.

Table 4 demonstrates the ability of our approach to obtain effective features; the table shows the top 10 features with high weights assigned for the Orthography data. An interesting aspect of the proposed method is that the process of the orthographic variants is interpretable through the feature weights.

Figure 2 shows plots of the F1 scores (y-axis) for the Inflection data when we change the number of active features (x-axis) by controlling the regularization parameter σ from 0.001 to 100. The larger the value we set for σ, the better the classifier performs, generally, with more active features. In extreme cases, the number of active features drops to 97 with $\sigma = 0.01$; nonetheless, the classifier still achieves 0.961 of the F1 score. The result suggests that a small set of substitution rules can accommodate most cases of inflectional variations.

3.3 Experiment 2: String transformation

The second experiment examined the performance of the string normalization tasks formalized in equation 1. In this task, a system was given a string s and was required to yield either its transformed form t^* ($s \neq t^*$) or the string s itself when the transformation is unnecessary for s. The conditional probability distribution (scorer) in equation 1 was modeled

System	Orthography			Derivation			Inflection			XTAG morph 1.5		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Morpha	.078	.012	.021	.233	.016	.029	.435	.682	.531	.830	.587	.688
CST's lemmatiser	.135	.160	.146	.378	.732	.499	.367	.762	.495	.584	.589	.587
Proposed method	**.859**	**.823**	**.841**	**.979**	**.981**	**.980**	**.973**	**.979**	**.976**	**.837**	**.816**	**.827**

Table 5: Performance of string transformation

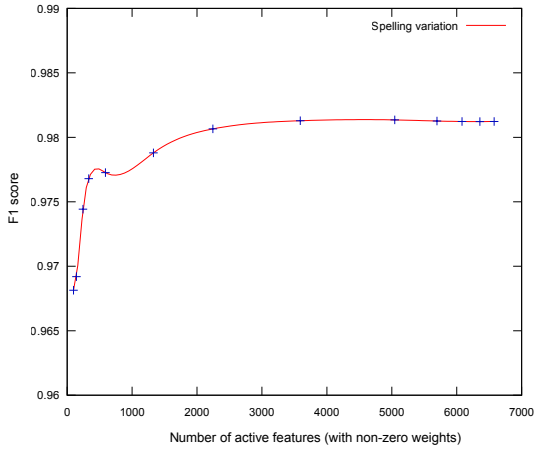

Figure 2: Number of active features and performance.

by the maximum entropy framework. Features for the maximum entropy model consist of: substitution rules between strings s and t, letter bigrams and trigrams in s, and letter bigrams and trigrams in t.

We prepared four datasets, *Orthography*, *Derivation*, *Inflection*, and *XTAG morphology*. Each dataset is a list of string pairs $\langle s, t \rangle$ that indicate the transformation of the string s into t. A source string s is identical to its destination string t when string s should not be changed. These instances correspond to the case where string s has already been lemmatized. For each string pair (s, t) in LRSPL[9], LRNOM, and LRAGR tables, we generated two instances $\langle s, t \rangle$ and $\langle t, t \rangle$. Consequently, a system is expected to leave the string t unchanged. We also used XTAG morphology[10] to perform a cross-domain evaluation of the lemmatizer trained on the Inflection dataset[11]. The entries in XTAG morphology that also appear in the Inflection dataset were 39,130 out of 317,322 (12.3 %). We evaluated the proposed method and CST's lemmatizer by performing ten-fold cross validation.

Table 5 reports the performance based on the number of correct transformations. The proposed method again outperformed the baseline systems with a wide margin. It is noteworthy that the proposed method can accommodate morphological inflections in the XTAG morphology corpus with no manual tuning or adaptation.

Although we introduced no assumptions about target tasks (e.g. a known vocabulary), the average number of positive substitution rules relevant to source strings was as small as 23.9 (in XTAG morphology data). Therefore, the candidate generator performed 23.9 substitution operations for a given string. It applied the decision rules (equation 7) 21.3 times, and generated 1.67 candidate strings per source string. The experimental results described herein demonstrated that the candidate generator was modeled successfully by the discriminative framework.

4 Related work

The task of string transformation has a long history in natural language processing and information retrieval. As described in Section 1, this task is related closely to various applications. Therefore, we specifically examine several prior studies that are relevant to this paper in terms of technical aspects.

Some researchers have reported the effectiveness of the discriminative framework of string similarity. MaCallum et al. (2005) proposed a method to train the costs of edit operations using Conditional Random Fields (CRFs). Bergsma and Kondrak (2007)

[9] We define that s precedes t in dictionary order.

[10] XTAG morphology database 1.5: `ftp://ftp.cis.upenn.edu/pub/xtag/morph-1.5/morph-1.5.tar.gz`

[11] We found that XTAG morphology contains numerous incorrect comparative and superlative adjectives, e.g., *unpopular* → *unpopularer* → *unpopularest* and *refundable* → *refundabler* → *refundablest*. Therefore, we removed inflection entries for comparative and superlative adjectives from the dataset.

presented an alignment-based discriminative string similarity. They extracted features from substring pairs that are consistent to a character-based alignment of two strings. Aramaki et al. (2008) also used features that express the different segments of the two strings. However, these studies are not suited for a candidate generator because the processes of string transformations are intractable in their discriminative models.

Dalianis and Jongejan (2006) presented a lemmatiser based on suffix rules. Although they proposed a method to obtain suffix rules from a training data, the method did not use counter-examples (negatives) for reducing incorrect string transformations. Tsuruoka et al. (2008) proposed a scoring method for discovering a list of normalization rules for dictionary look-ups. However, their objective was to transform given strings, so that strings (e.g., *studies* and *study*) referring to the same concept in the dictionary are mapped into the same string (e.g., *stud*); in contrast, this study maps strings into their destination strings that were specified by the training data.

5 Conclusion

We have presented a discriminative approach for generating candidates for string transformation. Unlike conventional spelling-correction tasks, this study did not assume a fixed set of destination strings (e.g. correct words), but could even generate unseen candidate strings. We used an L_1-regularized logistic regression model with substring-substitution features so that candidate strings for a given string can be enumerated using the efficient algorithm. The results of experiments described herein showed remarkable improvements and usefulness of the proposed approach in three tasks: normalization of orthographic variants, noun derivation, and lemmatization.

The method presented in this paper allows only one region of change in string transformation. A natural extension of this study is to handle multiple regions of changes for morphologically rich languages (e.g. German) and to handle changes at the phrase/term level (e.g., "estrogen receptor" and "receptor of oestrogen"). Another direction would be to incorporate the methodologies for semi-supervised machine learning to accommodate situations in which positive instances and/or unlabeled strings are insufficient.

Acknowledgments

This work was partially supported by Grants-in-Aid for Scientific Research on Priority Areas (MEXT, Japan), and for Solution-Oriented Research for Science and Technology (JST, Japan).

References

George W. Adamson and Jillian Boreham. 1974. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information Storage and Retrieval*, 10(7-8):253–260.

Farooq Ahmad and Grzegorz Kondrak. 2005. Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP 2005)*, pages 955–962.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L_1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, pages 33–40.

Eiji Aramaki, Takeshi Imai, Kengo Miyo, and Kazuhiko Ohe. 2008. Orthographic disambiguation incorporating transliterated probability. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008)*, pages 48–55.

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Shane Bergsma and Grzegorz Kondrak. 2007. Alignment-based discriminative string similarity. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL 2007)*, pages 656–663.

Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2003)*, pages 39–48.

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

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: a case study

in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565.

Andrew Carlson and Ian Fette. 2007. Memory-based context-sensitive spelling correction at web scale. In *Proceedings of the Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 166–171.

Qing Chen, Mu Li, and Ming Zhou. 2007. Improving query spelling correction using web search results. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pages 181–189.

Hercules Dalianis and Bart Jongejan. 2006. Handcrafted versus machine-learned inflectional rules: The euroling-siteseeker stemmer and cst's lemmatiser. In *In Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 663–666.

Okan Kolak and Philip Resnik. 2002. OCR error correction using a noisy channel model. In *Proceedings of the second international conference on Human Language Technology Research (HLT 2002)*, pages 257–262.

Grzegorz Kondrak. 2005. Cognates and word alignment in bitexts. In *Proceedings of the Tenth Machine Translation Summit (MT Summit X)*, pages 305–312.

Lillian Lee. 1999. Measures of distributional similarity. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL 1999)*, pages 25–32.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Mu Li, Yang Zhang, Muhua Zhu, and Ming Zhou. 2006. Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (Coling-ACL 2006)*, pages 1025–1032.

Andrew McCallum, Kedar Bellare, and Fernando Pereira. 2005. A conditional random field for discriminatively-trained finite-state string edit distance. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, pages 388–395.

I. Dan Melamed. 1999. Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25(1):107–130.

Guido Minnen, John Carroll, and Darren Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–223.

Gonzalo Navarro. 2001. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1):31–88.

Andrew Y. Ng. 2004. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning (ICML 2004)*, pages 78–85.

Martin F. Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

Yoshimasa Tsuruoka, John McNaught, and Sophia Ananiadou. 2008. Normalizing biomedical terms by minimizing ambiguity and variability. *BMC Bioinformatics*, Suppl 3(9):S2.

W. John Wilbur, Won Kim, and Natalie Xie. 2006. Spelling correction in the PubMed search engine. *Information Retrieval*, 9(5):543–564.