Alignment-HMM-based Extraction of Abbreviations from Biomedical Text

Dana Movshovitz-Attias
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
dma@cs.cmu.edu

William W. Cohen
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
wcohen@cs.cmu.edu

Abstract

We present an algorithm for extracting abbreviation definitions from biomedical text. Our approach is based on an alignment HMM, matching abbreviations and their definitions. We report 98% precision and 93% recall on a standard data set, and 95% precision and 91% recall on an additional test set. Our results show an improvement over previously reported methods and our model has several advantages. Our model: (1) is simpler and faster than a comparable alignment-based abbreviation extractor; (2) is naturally generalizable to specific types of abbreviations, *e.g.*, abbreviations of chemical formulas; (3) is trained on a set of unlabeled examples; and (4) associates a probability with each predicted definition. Using the abbreviation alignment model we were able to extract over 1.4 million abbreviations from a corpus of 200K full-text PubMed papers, including 455,844 unique definitions.

1 Introduction

Abbreviations and acronyms are commonly used in the biomedical literature for names of genes, diseases and more (Ambrus, 1987). Abbreviation definitions are a source of ambiguity since they may change depending on the context. The ability to recognize and extract abbreviations and map them to a full definition can be useful for Information Extraction tasks (Yu et al., 2007) and for the complete understanding of scientific biomedical text.

Yu et al. (2002) distinguish the two following uses of abbreviations: (1) *Common* abbreviations are those that have become widely accepted as synonyms, such as ⟨DNA, deoxyribonucleic acid⟩ or ⟨AIDS, acquired immunodeficiency syndrome⟩. These represent common fundamental and important terms and are often used, although not explicitly defined within the text (Fred and Cheng, 2003). In contrast, (2) *Dynamic* abbreviations, are defined by the author and used within a particular article. Such definitions can often overlap, depending on the context. For example, the term PBS most commonly abbreviates *Phosphate Buffered Saline*, but in other contexts may refer to the following: Pain Behavior Scale, Painful Bladder Syndrome, Paired Domain-Binding Site, Particle Based Simulation, Partitioned Bremer Support, Pharmaceutical Benefits Scheme, and more. Some abbreviations fall between these two definitions in the sense that they are normally defined in the text, however, they have become widely used, and therefore they do not normally overlap with other abbreviations. An example for this is the term ATP which, almost exclusively, abbreviates *adenosine triphosphate*, and is only rarely used in different contexts in biomedicine.

Gaudan et al. (2005) define two similar concepts, distinguishing *Global* and *Local* abbreviations. Global abbreviations are not defined within the document, similar to common abbreviation. Local abbreviations appear in the document alongside the long form, similar to dynamic abbreviations. The contextual ambiguity of dynamic, or local, abbreviations makes them an important target for abbreviation recognition tasks.

There is a great deal of variation in the way that different authors produce abbreviations. Our definition of *abbreviation* is quite flexible and can best be

Proceedings of the 2012 Workshop on Biomedical Natural Language Processing (BioNLP 2012), pages 47–55,
Montréal, Canada, June 8, 2012. ©2012 Association for Computational Linguistics

represented by the set of examples described in Table 1. These include simple acronyms, in which the first letter of every word from the long form is represented in the short form, as well as more complex cases such as: inner letter matches, missing short form characters, and specific substitutions (such as of a chemical element and its symbol). We generally assume that the abbreviated form contains some contraction of words or phrases from the full form. This definition is consistent with the one defined by many other extraction systems (see *e.g.*, (Schwartz and Hearst, 2002) and (Chang et al., 2002)).

We describe a method for extracting dynamic abbreviations, which are explicitly defined in biomedical abstracts. For each of the input texts, the task is to identify and extract ⟨*short form*, *long form*⟩ pairs of the abbreviations defined within the text. We also provide a mapping, formed as an alignment, between the characters of the two forms, and the probability of this alignment according to our model.

Our approach is based on dividing the abbreviation recognition task into the following stages: (1) Parsing the text and extracting *candidate* abbreviation pairs (long and short forms) based on textual cues, such as parentheses; (2) Recovering a valid alignment between the short and long form candidates (valid alignments are defined in Section 3.2). We perform a sequential alignment based on a pair-HMM; (3) Extracting a final short and long form from the alignment.

We will show that our approach is fast and accurate: we report 98% precision and 93% recall on a standard data set, and 95% precision and 91% recall on a validation set. The alignment model: (1) is simpler and faster than a comparable alignment-based abbreviation extractor; (2) is naturally generalizable to specific types of abbreviations; (3) is trained on a set of unlabeled examples; and (4) associates a probability with each predicted definition.

2 Related Work

A wide variety of methods have been introduced for recognizing abbreviations in biomedical context. Many utilize one of the following techniques: rule-based extraction, and extraction that relies on an alignment of the abbreviation and full definition. Abbreviation extraction methods have been used in two main contexts: to create online collections of abbreviations, normally extracted from PubMed abstracts (Zhou et al., 2006; Gaudan et al., 2005; Adar, 2004), and as part of larger learning frameworks, mainly for feature generation (Chowdhury et al., 2010; Huang et al., 2011).

Rule based extraction systems use a set of manually crafted pattern-matching rules to recognize and extract the pair of abbreviation and definition: Acrophile (Larkey et al., 2000) is an acronym recognition system that exploits morphological rules based on the case of the characters in the definitions. Unlike many of the other available systems, it recognized acronyms that are defined without parentheses; The Alice system (Ao and Takagi, 2005) is based on three extraction phases, each employing an elaborate set of over 15 rules, patterns and stop word lists. Liu and Friedman (2003) use a set of statistical rules to resolve cases in which an abbreviation is defined more than once with several different definitions. While these methods normally achieve high performance results, their main drawback is that they are difficult to implement and to extend. Rule development is normally based on a thorough investigation of the range of targeted abbreviations and the resulting heuristic patterns contain subtleties that are hard to recreate or modify.

Several extraction methods have been developed based on some variant of the Longest Common Subsequence algorithm (LCS) (Schwartz and Hearst, 2002; Chang et al., 2002; Taghva and Gilbreth, 1999; Bowden et al., 1997). These systems search for at least one possible alignment of an abbreviation and a full form definition.

The most widely used abbreviation extraction system is that presented by Schwartz and Hearst (2002). Their method scans the input text and extract pairs of candidate abbreviations from text surrounding parentheses. The algorithm scans the candidate definition from right to left, and searches for an implicit alignment of the definition and abbreviation based on few ad-hoc rules. This algorithm presents several constraints on the type of recognized abbreviations, the most restrictive being that every letter of the abbreviation must be matched during the process of scanning the definition. Of the variety of available extraction systems, this remains a popular choice due to its simplicity and speed. However, as the au-

Short	Long	Type of Abbreviation
AMS	Associated Medical Services	Acronym using the first letter of each long-form word.
PS	postsynaptic	Inner letters are represented in the abbreviation.
NTx	cross-linked N-telopeptides	1. Phonetic substitution (*cross* → *x*). 2. The short form is out-of-order. 3. Words from the long form are missing in the short form (*linked*).
EDI-2	Eating Disorders Inventory	Characters from the short form are missing in the long form (*-2*).
NaB	sodium butyrate	Substitution of a chemical element by its symbol (*sodium* → *Na*).
MTIC	5-(3-N-methyltriazen-1-yl)-imidazole-4-carboxamide	Chemical formula.
EBNA-1	Epstein-Barr virus (EBV) nuclear antigen 1	Recursive definition, in which the long form contains another abbreviation definition.
3-D	three-dimensional	Substitution of a number name and symbol (*three* → *3*).
A&E	accident and emergency	Substitution of a word and symbol (*and* → &).
anti-Tac	antibody to the alpha subunit of the IL-2 receptor	Synonym: the short form commonly represents the long form, although it is not a direct abbreviation of it.
R.E.A.L.	'Revised European-American Classification of Lymphoid Neoplasms'	The long- and/or short-forms contain characters that are not directly related to the abbreviation (*e.g.*, punctuation symbols).

Table 1: Examples of biomedical abbreviations.

thors report, this algorithm is less specific than other approaches and consequently results in lower recall. We will show that by performing an explicit alignment of the abbreviation using an alignment-HMM, our model results in more accurate predictions, and that the edit operations used in the alignment allow for natural extensions of the abbreviations domain.

Another frequently used alignment based approach is that of Chang *et al.* (2002), and it is closest to our approach. After calculating an abbreviation alignment, they convert the set of aligned terms into a feature vector which is scored using a binary logistic regression classifier. Using a correct threshold on the alignment scores produces a high performance abbreviation extractor. However this approach has several drawbacks. The run-time of this algorithm is fairly long (see Section 4.3), in part due to the steps following the alignment recovery, *i.e.*, calculating a feature vector, and generating an alignment score. Additionally, choosing a score threshold may depend on the genre of text, and different thresholds lead to a variety of quality in the results. We will show that presenting limitations on the range of available alignments can produce correct alignments more efficiently and quickly, maintaining high quality results, without the need for threshold selection. Our alignment method distinguishes and penalizes inner and leading gaps in the alignment, and it applies a set of constraints on the range of legal alignments. We will also show that relying solely on constrained alignments still allows for flexibility in the definition of the range of desired abbreviations.

Ristad and Yianilos (1998) proposed a single state alignment-HMM for learning string-edit distance based on matched strings. In later work, Bilenko and Mooney (2003) extend this model to include affine gaps, by including in their model separate states for Matches, Deletions and Insertions. McCallum et al. (2005) describe a discriminative string edit CRF, following a similar approach to that of Bilenko and Mooney. The CRF model includes two disjoint sets of states, each representing either "matching" or "mismatching" string pairs. Each of the sets is similar to the model described by Bilenko and Mooney. All of these models require labeled training examples, and the CRF approach also requires negative training examples, which train the "mismatching" states of the model. We describe an alignment HMM that is suited for aligning abbreviation long and short forms, and does not require any labeling of the input text or training examples.

3 Method

In the following sections we describe a method for extracting *candidate* abbreviation definitions from text, and an alignment model with affine gaps for

Description	Result												
i. Input sentence:	“anti-sperm antibodies were studied by indirect mixed anti-globulin reaction test (MAR)”												
ii. Candidate:	⟨MAR, by indirect mixed anti-globulin reaction test⟩												
iii. Alignment:													
HMM States	LG	LG	LG	LG	M	M	M	M	IG	M	M	M	IG
Short Form					M		A				R		
Long Form	by		indirect		mixed		anti	-	globulin		reaction		test
iv. Abbreviation:	⟨MAR, mixed anti-globulin reaction test⟩												

Table 2: Example of the processing steps of a sample sentence. (i) Input sentence containing a single abbreviation. (ii) Candidate ⟨short form, long form⟩ pair extracted from the sentence (after truncating the long-form). (iii) The most likely (Viterbi) alignment of the candidate pair, using our alignment model. Each state corresponds to a single edit-operation, which consumed the corresponding short-form and long-form characters in the alignment. (iv) Final abbreviation, extracted from the alignment by removing leading gaps.

matching the two forms of a candidate definition. Finally we describe how to extract the final abbreviation prediction out of the alignment.

3.1 Extracting candidate abbreviations

The process described below scans the text for textual cues and extracts a list of candidate abbreviation pairs, for every input document, in the form: ⟨*short form*, *long form*⟩. The following text also describes the restrictions and conditions of what we consider to be valid candidate pairs. The assumptions made in this work are generally less restrictive that those introduced by previous extraction systems and they lead to a larger pool of candidate definitions. We will later show that false candidates normally produce invalid alignment of their short and long forms, according to our alignment model, and so they are removed and do not affect the final results.

The parsing process includes a search for both single abbreviations, and abbreviation patterns. An example of a sentence with a single abbreviation can be seen in Table 2(i). We consider the following two cases of a single abbreviation definition: (1) “long form (short form)”, and (2) “short form (long form)”. Note that in some cases, the term within the parenthesis is parsed, *e.g.*, in the following text, *ELISA* is extracted from the parenthesis, by removing the text beyond the ’;’ symbol: “. . . human commercial enzyme-linked immunosorbent assay (ELISA; BioGen, Germany) . . . ”.

We also consider *abbreviation patterns* which define multiple abbreviations simultaneously, as demonstrated by these examples:

- “anti-sperm (ASA), anti-phospholipid (APA), and antizonal (AZA) antibodies” – The main noun (*antibodies*) follows the pattern.
- “Epithelial-mesenchymal transition (EMT) and interaction (EMI)” – The main noun (*Epithelial-mesenchymal*) is at the head of the pattern.

Using textual cues (patterns and parentheses) we extract candidate short and long forms. Whenever possible, we consider the term within the parenthesis as the short form, and the text to the left of the parenthesis (until the beginning of the sentence) as the candidate long form. We consider valid short forms to be no longer than 3 words, having between 1 and 15 characters, and containing at least one letter. In the case that the candidate short form was found to be invalid by these definitions, we switch the assignment of long and short forms. The long-form string is truncated, following Park and Byrd (2001), to a length of $\min(|A| + 5, |A| * 2)$, where $|A|$ is the length of the short form.

The length of the candidate long form is estimated using the Park and Byrd formula, and it is therefore normally the case that the resulting candidate long form contains some leading characters that are not part of the abbreviation definition. Next, we define an alignment between short and long form strings

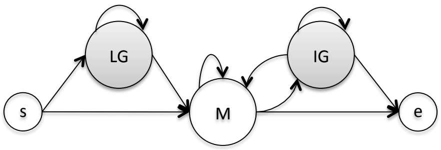

Figure 1: Abbreviation alignment HMM model with states: start (s), leading gaps (LG), match (M), inner gap (IG) and end (e).

Edit Operation	SF Match	LF Match	Valid States
LF deletion	ϵ	alpha-numeric char	LG, IG
LF deletion	ϵ	punct. symbol	LG, M
LF deletion	ϵ	word	LG, IG
SF deletion	digit or punct.	ϵ	IG
Match	char	(partial) word	M
Match	char	char	M
Substitution	'&'	'and'	M
Substitution	'1'-'9'	'one'-'nine'	M
Substitution	chem. symbol	chemical name	M

Table 3: Edit operations used in the alignment HMM model including, long form (LF) and short form (SF) deletions, matches and substitutions. We note the SF and LF characters consumed by each edit operation, and the HMM states in which it may be used.

which detects possible segments that are missing in the alignment in either string (gaps).

3.2 Aligning candidate long and short forms

For each of the candidate pairs produced in the previous step, we find the best alignment (if any) between the short and the long form strings. We describe an alignment HMM that is suited for abbreviation alignments. The model is shown in Figure 1, and Table 2 shows the parsing process of a sample sentence, including an alignment created for this sample using the model.

3.3 Abbreviation Alignment with Affine Leading and Inner Gaps

An alignment between a long and a short form of an abbreviation can be modeled as a series of edit operations between the two strings, in which characters from the short form may match a single or a series of characters from the long form. In previous work, Bilenko and Mooney (2003) describe a generative model for string edit distance with affine gaps, and an Expectation Maximization algorithm for learning the model parameters using a labeled set of matching strings. We propose a similar model for aligning the short and long form of an abbreviation, using an affine cost model for gaps

$$cost(g) = s + e \cdot l \tag{1}$$

where s is the cost of starting a gap, e is the cost of extending a gap and l is the length of the gap. In our method, we use extracted candidate pairs (candidate short and long forms) as training examples.

As described above, candidate long forms are formed by extracting text preceding parentheses and truncating it to some length. This process may lead to candidate long forms that contain leading characters that do not belong to the abbreviation, which will result in leading gaps in the final alignment. For example, the candidate long form presented in Table 2(ii) contains the leading text "by indirect ". While extra leading text is expected as an artifact of our candidates extraction method, inner alignment gaps are not expected to commonly appear in abbreviation alignments, and are usually an indication of a bad alignment. The example presented in Table 2 is of an abbreviation that *does* contain inner gaps (*e.g.*, *globulin*) despite being a valid definition.

We distinguish leading and inner alignment gaps using a model with five states: Leading Gap (LG), Match (M), Inner Gap (IG), and two "dummy" states for the beginning and end of an alignment (Figure 1). Since leading and inner gaps are represented by different states, their penalization is not coupled, *i.e.*, they are associated with different s, e and l costs. We use the EM algorithm to learn the model parameters, based on a set of unlabeled candidate pairs, following the assumption that many false-candidates will not produce a valid alignment, and will not affect training. This is in contrast to previous string edit distance models, which require labeled training examples.

The main effort in developing a successful abbreviation alignment model involves generating a meaningful set of edit operations. The edit operations used in our model, $E = E_d \cup E_m \cup E_s$, is shown in Table 3 and includes: E_d, deletions of characters or words from the long form, or of single characters from the short form; E_m, matches of a full of partial word from the long form to a character in the short form; and E_s, word substitutions in which a word from the long form is replaced by a symbol in the short form. Note that: (1) while all types of deletions from the long form are valid, deletions from the short form are limited to digits and punctuation symbols, and (2) deletion of non-alpha-numeric characters from the long form is not considered as opening a gap but as a match, as it is common for non-alphanumeric characters to be missing in an abbreviation (*i.e.*, be "matched" with the empty string, ϵ).

Let $x = x_1 \dots x_T$ be the short form candidate, $y = y_1 \dots y_V$ be the long form candidate, and $a = \langle a_p \rangle_{p=1}^{n}$, $a_p = (e_p, q_p, ix_p, jy_p)$, be a possible alignment of the strings x and y. a represents as a sequence of HMM transitions, a_p, where $e_p \in E$ is an edit operation that consumes characters from x (deletion from the long form), y (deletion from the short form), or both (match or substitution), up to position ix_p in x and jy_p in y, and is associated with a transition in the model to state $q_p \in \{\mathrm{LG}, \mathrm{M}, \mathrm{IG}, \mathrm{e}\}$. Let $\pi(q, q')$ be the transition probability between states q and q', and let $\tau(q, e)$ be the emission probability of the edit operation e at state q. Given a candidate abbreviation pair $\langle x, y \rangle$, and the model parameters π and τ, the probability of an alignment is given by

$$p(a|x, y, \pi, \tau) = \prod_{p=1}^{|a|} \pi(q_{p-1}, q_p) \cdot \tau(q_p, e_p) \quad (2)$$

where q_0 is the start state. This probability can be calculated efficiently using dynamic programming with the forward-backward algorithm, and the most likely alignment corresponds to the Viterbi distance between x and y.

In our method, the model parameters, π and τ, are estimated using the EM algorithm on an unlabeled training set of candidate pairs that have been extracted from the text, without any further processing. At each EM iteration, we train on pairs that have valid alignments (see below) with non-zero probability under the model parameters at that iteration.

3.3.1 Valid Alignments

Given the edit operations defined above, the only valid way of matching a letter from the short form to the long form is by matching that letter to the beginning of a full or partial word, or by matching that letter using a substitution operation. There is no edit operation for deleting letters from the short form (only digits and punctuation symbols can be deleted). This means that for some candidate pairs there are no valid alignments under this model, in which case, no abbreviation will be predicted.

3.3.2 Extracting the Final Abbreviation

Given a valid alignment a between the candidate pair, x and y, we create a truncated alignment, a', by removing from a initial transitions in which $q_p = \mathrm{LG}$. We consider a' valid if the number of matches in $a' = \langle a'_p \rangle_{p=1}^{n'}$ is greater than the number of deletions,

$$\sum_{p=1}^{n'} I(q'_p = \mathrm{M}) > \sum_{p=1}^{n'} I(q'_p = \mathrm{IG}) \quad (3)$$

where I is an indicator function.

The final abbreviation prediction is given by the portions of the x and y strings that are associated with a', named x' and y', respectively. These may be truncated compared to x and y, as leading alignment gaps are removed. The final alignment probability is given by $p(a'|x', y', \pi, \tau)$.

3.4 Substitution Edit Operations

In contrast to rule-based extraction algorithms, in our model, it is easy to introduce new types of edit operations, and adjust the model to recognize a variety of abbreviation types. As an example, we have added a number of substitution operations (see Table 3), including an operation for the commonly used convention of replacing a chemical element name (*e.g.*, *Sodium*) with its symbol (*Na*). These types of operations are not available using simpler models, such as that presented by Schwartz and Hearst (2002), making it impossible to recognize some important biomedical entities, such as chemical compounds (*e.g.*, $\langle NaB, SodiumButyrate \rangle$). In contrast, such additions are natural in our model.

4 Evaluation

4.1 Abbreviation Extraction Analysis

We evaluated the alignment abbreviation model over two data sets (Table 4). The method was tuned using

Data Set	Name	Abstracts	Abbreviations	Testing Method
Development (D)	Medstract	400	483	10-fold cross validation.
Validation (V)	PubMed Sample	50	76	Training on set D and testing on set V.

Table 4: Evaluation Data Sets.

Model	D (average %)			V (%)		
	P	R	F1	P	R	F1
Alignment HMM	98	**93**	**96**	95	91	**93**
SH	96	88	91	**97**	83	89
Chang $_{0.88}$	**99**	46	62	**97**	47	64
Chang $_{0.14}$	94	89	91	95	91	**93**
Chang $_{0.03}$	92	91	91	88	**93**	90
Chang $_{0}$	49	92	64	53	**93**	67

Table 5: Results on validation (V) and development (D) sets. Average results are shown for D set, which was tested using 10-fold cross-validation (results rounded to nearest percent, all standard deviations were < 0.1)

10 fold cross-validation over the publicly available Medstract corpus (Pustejovsky et al., 2002) which includes 400 Medline abstracts. The online version of the corpus was missing the Gold Standard annotations throughout the development of our algorithm, nor was it possible to get them through communication with the authors. We therefore hand-annotated the Medstract data, yielding 483 abbreviation definitions in the form of ⟨*short form*, *long form*⟩ pairs. In order to be consistent with previous evaluations over Medstract, our annotations include only definitions in which either the short or the long form appear in parenthesis, and it is assumed that there are no trailing gaps in the term preceding the parenthesis, although our model does detect such gaps.

We compare our results with two algorithms available for download: the Schwartz and Hearst (SH; (2002)) algorithm[1], and the Chang *et al.* (2002) algorithm[2] used at three score cutoffs reported in their paper (0.88, 0.14, 0.03). We also use a fourth score cutoff of 0 to account for any legal alignments produced by the Chang model.

In Table 5 we report precision (P), recall (R) and F1 scores for all methods, calculated by

$$P = \frac{\text{correct predicted abbreviations}}{\text{all predicted abbreviations}} \quad (4)$$

$$R = \frac{\text{correct predicted abbreviations}}{\text{all correct abbreviations}} \quad (5)$$

On the development set, our alignment model achieves 98% precision, 93% recall and 96% F1 (average values over cross-validation iterations, with standard deviations all under 0.03).

To test the final model we used a validation dataset consisting of 50 abstracts, randomly selected out of a corpus of 200K full-text biomedical articles taken from the PubMed Central Open Access Subset (extracted in October 2010)[3]. These were hand-annotated, yielding 76 abbreviation definitions.

On the validation set, we predicted 69 out of 76 abbreviations, with 4 false predictions, giving 95% precision, 91% recall and 93% F1. Our alignment model results in higher F1 score over all baselines in both datasets (with Chang$_{0.14}$ giving equal results on the validation set). Our results are most comparable with the Chang model at a score cutoff of 0.14, though our model does not require selecting a score cutoff, and as we will show, it is considerably faster. Interestingly, our model results in lower recall than precision on both data sets. This may be due to a limited scope of edit operations.

In order to evaluate the usability of our method, we used it to scan the 200K full-text documents of the PubMed Central Open Access Subset corpus. The process completed in under 3 hours, yielding over 1.4 million abbreviations, including 455,844 unique definitions. A random sample of the extracted abbreviations suggests a low rate of false positive predictions.

4.2 Error Analysis

Our model makes 4 incorrect predictions on the validation set, 3 of which are partial matches to the

[1] Taken from http://biotext.berkeley.edu/software.html

[2] Taken from http://abbreviation.stanford.edu

[3] http://www.ncbi.nlm.nih.gov/pmc/

Description	D	V
Letters in short form are missing (*e.g.*, ⟨GlyRalpha2, glycine alpha2⟩)	5	3
Abbreviation missed due to extraction rules.	6	1
Abbreviation is a synonym (*e.g.*, ⟨IRX-2, natural cytokine mixture⟩)	5	1
Abbreviation letters are out-of-order (*e.g.*, ⟨VSV-G, G glycoprotein of vesicular stomatitis virus⟩)	4	1
Correct alignment was found but it is invalid due to many inner gaps (see Section 3.3.1).	5	0
Abbreviations of chemical formulas or compounds.	4	0

Table 6: Abbreviations missed in development (D) and validation (V) sets.

correct definitions, *e.g.*, we predict the pair ⟨GlOx, glutamate oxidase⟩ instead of ⟨GlOx, L-glutamate oxidase⟩. On the development set, 3 out of 5 incorrect predictions are partial matches.

Our model did not extract 7 of the abbreviations from the validation set and 33 from the development set. Many of these abbreviations (6 from the validation set and 29 from the development set) had one of the properties described in Table 6. The remaining 5 definitions have been missed due to miscellaneous issues. Note that while we added several substitution operations for chemical formula recognition, the elaborate set of operations required for recovering the full range of chemical formulas was not included in this work, leading to 4 chemical formula abbreviations being missed.

4.3 Run-Time Analysis

We provide an estimated comparison of the run time of our method and the baseline algorithms. This analysis is especially interesting for cases in which an abbreviation extraction model is included within a larger learning framework (Chowdhury et al., 2010; Huang et al., 2011), and may be used in it in an online fashion. Run time was evaluated on an Apple iMac with 4GB 1333 MHz RAM, and a 3.06 GHz Core i3, double-core processor, by running all models on a random set of 400 abstracts. In order to evaluate the run time contribution of the substitution operations introduced in our model we ran it both with (88 $\frac{\text{docs}}{\text{sec}}$) and without (98 $\frac{\text{docs}}{\text{sec}}$) the use of substitution operations. We find that using substitutions did not have considerable effect on run time, adding under 1 ms for processing each document. We should note that the performance of the substitution-less model on this test data was similar to that of the original model, as substitutions were relevant to only a smaller portion of the abbreviations. As expected, the SH algorithm is considerably faster (6451 $\frac{\text{docs}}{\text{sec}}$) than our model, as it is based on only a number of simple rules. The Chang model, however, is slower (4 $\frac{\text{docs}}{\text{sec}}$) as it includes processing steps following the discovery of an abbreviation alignment, which means that our model provides comparable results to the Chang model and runs an order-of-magnitude faster.

5 Conclusions and Discussion

We presented a method for extracting abbreviation definitions with high precision and high recall (95% precision, 91% recall and 93% F1 on a validation set). Our model achieves higher F1 on both the development and validation data sets, when compared with two popular extraction methods.

Our approach is based on a sequential generative model, aligning the short and long form of an abbreviation. Using the proposed method we extracted 1.4 million abbreviations from a corpus of 200K PubMed articles. This data can be valuable for Information Extraction tasks and for the full understanding of biomedical scientific data.

The alignment abbreviation extractor can be easily extended by adding edit-operations over short and long forms. This was demonstrated by including substitutions of chemical elements and their symbols, which facilitates recognition of chemical formulas and compounds.

We have identified the main classes of abbreviation definitions missed by our approach. These include out-of-order matches, synonym-like abbreviations, and short forms with excess letters. It may be possible to address some of these issues by including “global” information on abbreviations, such as the occurrence of frequent definitions.

Acknowledgments

This work was funded by grant 1R101GM081293 from NIH, IIS-0811562 from NSF and by a gift from Google. The opinions expressed in this paper are solely those of the authors.

References

E. Adar. 2004. Sarad: A simple and robust abbreviation dictionary. *Bioinformatics*, 20(4):527–533.

JL Ambrus. 1987. Acronyms and abbreviations. *Journal of medicine*, 18(3-4):134.

H. Ao and T. Takagi. 2005. Alice: an algorithm to extract abbreviations from medline. *Journal of the American Medical Informatics Association*, 12(5):576–586.

M. Bilenko and R.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*, pages 39–48. ACM.

P.R. Bowden, P. Halstead, and T.G. Rose. 1997. Dictionaryless english plural noun singularisation using a corpus-based list of irregular forms. *LANGUAGE AND COMPUTERS*, 20:339–352.

J.T. Chang, H. Schütze, and R.B. Altman. 2002. Creating an online dictionary of abbreviations from medline. *Journal of the American Medical Informatics Association*, 9(6):612–620.

M. Chowdhury, M. Faisal, et al. 2010. Disease mention recognition with specific features. In *Proceedings of the 2010 Workshop on Biomedical Natural Language Processing*, pages 83–90. Association for Computational Linguistics.

H.L. Fred and T.O. Cheng. 2003. Acronymesis: the exploding misuse of acronyms. *Texas Heart Institute Journal*, 30(4):255.

S. Gaudan, H. Kirsch, and D. Rebholz-Schuhmann. 2005. Resolving abbreviations to their senses in medline. *Bioinformatics*, 21(18):3658–3664.

M. Huang, J. Liu, and X. Zhu. 2011. Genetukit: a software for document-level gene normalization. *Bioinformatics*, 27(7):1032–1033.

L.S. Larkey, P. Ogilvie, M.A. Price, and B. Tamilio. 2000. Acrophile: an automated acronym extractor and server. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 205–214. ACM.

H. Liu, C. Friedman, et al. 2003. Mining terminological knowledge in large biomedical corpora. In *Pac Symp Biocomput*, pages 415–426.

A. McCallum, K. Bellare, and F. Pereira. 2005. A conditional random field for discriminatively-trained finite-state string edit distance. In *Conference on Uncertainty in AI (UAI)*.

Y. Park and R.J. Byrd. 2001. Hybrid text mining for finding abbreviations and their definitions. In *Proceedings of the 2001 conference on empirical methods in natural language processing*, pages 126–133.

J. Pustejovsky, J. Castano, R. Sauri, A. Rumshinsky, J. Zhang, and W. Luo. 2002. Medstract: creating large-scale information servers for biomedical libraries. In *Proceedings of the ACL-02 workshop on Natural language processing in the biomedical domain-Volume 3*, pages 85–92. Association for Computational Linguistics.

E.S. Ristad and P.N. Yianilos. 1998. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532.

A.S. Schwartz and M.A. Hearst. 2002. A simple algorithm for identifying abbreviation definitions in biomedical text. In *Pacific Symposium on Biocomputing 2003: Kauai, Hawaii, 3-7 January 2003*, page 451. World Scientific Pub Co Inc.

K. Taghva and J. Gilbreth. 1999. Recognizing acronyms and their definitions. *International Journal on Document Analysis and Recognition*, 1(4):191–198.

H. Yu, G. Hripcsak, and C. Friedman. 2002. Mapping abbreviations to full forms in biomedical articles. *Journal of the American Medical Informatics Association*, 9(3):262–272.

H. Yu, W. Kim, V. Hatzivassiloglou, and W.J. Wilbur. 2007. Using medline as a knowledge source for disambiguating abbreviations and acronyms in full-text biomedical journal articles. *Journal of biomedical informatics*, 40(2):150–159.

W. Zhou, V.I. Torvik, and N.R. Smalheiser. 2006. Adam: another database of abbreviations in medline. *Bioinformatics*, 22(22):2813.