Learning the Structure of Variable-Order CRFs: a Finite-State Perspective

Thomas Lavergne and François Yvon
LIMSI, CNRS, Univ. Paris-Sud, Université Paris Saclay
Campus Universitaire, F-91 403 Orsay, France
{lavergne, yvon}@limsi.fr

Abstract

The computational complexity of linear-chain Conditional Random Fields (CRFs) makes it difficult to deal with very large label sets and long range dependencies. Such situations are not rare and arise when dealing with morphologically rich languages or joint labelling tasks. We extend here recent proposals to consider variable order CRFs. Using an effective finite-state representation of variable-length dependencies, we propose new ways to perform feature selection at large scale and report experimental results where we outperform strong baselines on a tagging task.

1 Introduction

Conditional Random Fields (CRFs) (Lafferty et al., 2001; Sutton and McCallum, 2006) are a method of choice for many sequence labelling tasks such as Part of Speech (PoS) tagging, Text Chunking, or Named Entity Recognition. Linear-chain CRFs are easy to train by solving a convex optimization problem, can accommodate rich feature patterns, and enjoy polynomial exact inference procedures. They also deliver state-of-the-art performance for many tasks, sometimes surpassing seq2seq neural models (Schnober et al., 2016).

A major issue with CRFs is the complexity of training and inference procedures, which are quadratic in the number of possible output labels for first order models and grow exponentially when higher order dependencies are considered. This is problematic for tasks such as precise PoS tagging for Morphologically Rich Languages (MRLs), where the number of morphosyntactic labels is in the thousands (Hajič, 2000; Müller et al., 2013). Large label sets also naturally arise when joint labelling tasks (eg. simultaneous PoS tagging and text chunking) are considered. For such tasks, processing first-order models is demanding, and full size higher-order models are out of the question. Attempts to overcome this difficulty are based on a greedy approach which starts with first-order dependencies between labels and iteratively increases the scope of dependency patterns under the constraint that a high-order dependency is selected only if it extends an existing lower order feature (Müller et al., 2013). As a result, feature selection may only choose only few higher-order features, motivating the need for an effective variable-order CRF (voCRF) training procedure (Ye et al., 2009).\footnote{This is reminiscent of variable order HMMs, introduced eg. in (Schütze and Singer, 1994; Ron et al., 1996).}

The latest implementation of this idea (Vieira et al., 2016) relies on (structured) sparsity promoting regularization (Martins et al., 2011) and on finite-state techniques, handling high-order features at a small extra cost (see § 2). In this approach, the sparse set of label dependency patterns is represented in a finite-state automaton, which arises as the result of the feature selection process.

In this paper, we somehow reverse the perspective and consider VoCRF training mostly as an automaton inference problem. This leads us to consider alternative techniques for learning the finite-state machine representing the dependency structure of sparse VoCRFs (see § 3). Two lines of enquiries are explored: (a) to take into account the internal structure of large tag sets in order to learn better and/or leaner feature sets; (b) to detect unconditional structural dependencies in label sequences in order to speed-up the discovery of useful features. These ideas are implemented in 6 feature selection strategies, allowing us to explore a large set of dependency structures. Relying on lazy finite-state operations, we train VoCRFs up to order 5, and achieve PoS tagging performance that
surpass strong baselines for two MRLs (see § 4).

2 Variable order CRFs

In this section, we recall the basics of CRFs and VoCRFs and introduce some notations.

2.1 Basics

First-order CRFs use the following model:
\[
p_{\theta}(y|x) = Z_{\theta}(x)^{-1} \exp(\theta^T F(x, y)) \tag{1}
\]
where \( x = (x_1, \ldots, x_T) \) and \( y = (y_1, \ldots, y_T) \) are the input (in \( \mathcal{X}^T \)) and output (in \( \mathcal{Y}^T \)) sequences and \( Z_{\theta}(x) \) is a normalizer. Each component \( F_j(x, y) \) of the global feature vector decomposes as a sum of local features \( \sum_{t=1}^{T} f_j(y_{t-1}, y_t, x_t) \) and is associated to parameter \( \theta_j \). Local features typically use binary tests and take the form:
\[
f_{u,g}(y_{t-1}, y_t, x_t) = I(y_t = u \wedge g(x, t))
\]
\[
f_{w,g}(y_{t-1}, y_t, x_t) = I(y_{t-1}y_t = uw \wedge g(x, t))
\]
where \( I() \) is an indicator function and \( g() \) tests a local property of \( x \) around \( x_t \). In this setting, the number of parameters is \( |\mathcal{Y}|^2 \times |\mathcal{X}|_{\text{train}} \), where \( |A| \) is the cardinality of \( A \) and \( |\mathcal{X}|_{\text{train}} \) is the number of values of \( g(x, t) \) observed in the training set. Even in moderate size applications, the parameter set can be very large and contain dozens of millions of features, due to the introduction of sequential dependencies in the model.

Given \( N \) i.i.d. sequences \( \{x^{(i)}, y^{(i)}\}_{i=1}^{N} \), estimation is based on the minimization of the negated conditional log-likelihood \( l() \). Optimizing this objective requires to compute its gradient and to repeatedly evaluate the conditional expectation of the feature vector. This can be done using a forward-backward algorithm having a complexity that grows quadratically with \( |\mathcal{Y}| \). \( l() \) is usually complemented with a regularization term so as to avoid overfitting and stabilize the optimization. Common regularizers use the \( \ell_1 \)- or the \( \ell_2 \)-norm of the parameter vector, the former having the benefit to promote sparsity, thereby performing automatic feature selection (Tibshirani, 1996).

2.2 Variable order CRFs (VoCRFs)

When the label set is large, many pairs of labels never occur in the training data and the sparsity of label ngrams quickly increases with the order \( p \) of the model. In the variable order CRF model, it is assumed that only a small number of ngrams (out of \(|\mathcal{Y}|^p \)) are associated with a non-zero parameter value. Denoting \( \mathcal{W} \) the set of such ngrams and \( w \in \mathcal{W} \), a generic feature function is then \( f_{w,g}(w, x, t) = I(y_{t-s} \ldots y_t = w \wedge g(x, t)) \).

In (order-\( p \)) VoCRFs, the computational cost of training and inference is proportional to the size of a finite-state automaton \( A[\mathcal{W}] \) encoding the patterns in \( \mathcal{W} \), which can be much less than \(|\mathcal{Y}|^p \). Our procedure for building \( A[\mathcal{W}] \) is sketched in Algorithm 1, where TrieInsert inserts a string \( w \) in a trie, Pref(\( \mathcal{W} \)) computes the set of prefixes of the strings in \( \mathcal{W} \), \( \text{LgSuff}(v, \mathcal{U}) \) returns the longest suffix of \( v \) in \( \mathcal{U} \), and FailureTrans is a special \( \epsilon \)-transition used only when no labelled transition exists (Allauzen et al., 2003). Each state (or pattern prefix) \( v \) in \( A[\mathcal{W}] \) is associated with a set of feature functions \( \{f_{u,g} : u \in \text{Suff}(v, g)\} \). The forward step of the gradient computation maintains one value \( \alpha(v, t) \) per state and time step, which is recursively accumulated over all paths ending in \( v \) at time \( t \).

The next question is to identify \( \mathcal{W} \). The simplest method keeps all the ngrams viewed in training, additionally filtering rare patterns (Cuong et al., 2014). However, frequency based feature selection does not take interactions into account and is not the best solution. Ideally, one would like to train a complete order-\( p \) model with a sparsity promoting penalty, a technique that only works for small label sets.\(^6\) The greedy algorithm of

---

**Algorithm 1: Building \( A[\mathcal{W}] \)**

\[ \mathcal{W} : \text{list of patterns, } A[\mathcal{W}] \text{ initially empty} \]
\[ \mathcal{U} = \text{Pref}(\mathcal{W}) \]
\[ \text{foreach } w \in \mathcal{W} \text{ do} \]
\[ \quad \text{TrieInsert}(w, A[\mathcal{W}]) \]
\[ \quad \text{// Add missing transitions} \]
\[ \text{foreach } u = vy \in \mathcal{U} \text{ do} \]
\[ \quad \text{new FailureTrans}(u, \text{LgSuff}(v, \mathcal{U})) \]

---

\(^3\)More precisely, Vieira et al. (2016) consider \( \overline{\mathcal{W}} \), the closure of \( \mathcal{W} \) under suffix and last character substitution, which factors as \( \overline{\mathcal{W}} = \mathcal{H} \times \mathcal{Y} \). The complexity of training depends on the size of the finite-state automaton representing \( \overline{\mathcal{W}} \).

\(^4\)A trie has one state for each prefix.

\(^5\)This was also suggested by Cotterell and Eisner (2015) as a way to build a more compact pattern automaton.

\(^6\)Upon reaching a state \( v \), we need to access the features that fire for that pattern, and also for all its suffixes. Each state thus stores a set of pattern; each pattern is associated with a set of tests on the observation (cf. 2.1).

\(^7\)Recall that the size of parameter set is exponential wrt. the model order.
Schmidt and Murphy (2010); Vieira et al. (2016) is more scalable: it starts with all unigram patterns and iteratively grows \( \mathcal{W} \) by extending the ngrams that have been selected in the simpler model. At each round of training, feature selection is performed using a \( \ell_1 \) penalty and identifies the patterns that will be further augmented.

3 Learning patterns

We introduce now several alternatives for learning \( \mathcal{W} \). Our motivation for doing so is twofold: (a) to take the internal structure of large label sets into account; (b) to identify more abstract patterns in label sequences, possibly containing gaps or iterations, which could yield smaller \( A[\mathcal{W}] \). As discussed below, both motivations can be combined.

3.1 Greedy \( \ell_1 \)

The greedy strategy iteratively grows patterns up to order \( p \). Considering all possible unigram and bigram patterns, we train a sparse model to select a first set of useful bigrams. In subsequent iterations, each pattern \( w \) selected at order \( k \) is extended in all possible ways to specify the pattern set at order \( k + 1 \), which will be filtered during the next training round. This approach is close, yet simpler, than the group lasso approach of Vieira et al. (2016) and experimentally yields slightly smaller pattern sets (see Table 2). This is because we do not enforce closure under last-character replacement: once pattern \( w \) is pruned, longer patterns ending in \( w \) are never considered.\(^7\)

3.2 Component-wise training

Large tag sets often occur in joint tasks, where multiple levels of information are encoded in one compound tag. For instance, the fine grain labels in the Tiger corpus (Brants et al., 2002) combine PoS and morphological information in tags such as \( \text{NN.Dat.Sg.Fem} \) for a feminine singular dative noun. In the sequel, we refer to each piece of information as a tag component. We assume that all tags contain the same components, using a “non-applicable” value whenever needed. Using features that test arbitrary combinations of tag components would make feature selection much more difficult, as the number of possible patterns grows combinatorially with the number of components. We keep things simple by allowing features to only evaluate one single component at a time: this allows us to identify dependencies of different orders for each component.

Assuming that each tag \( y \) contains \( K \) components \( y = [z_1, z_2, \ldots, z_K] \), with \( z_k \in \mathcal{Y}_k \), \( \mathcal{W} \) is then computed as in § 3.1, except that we now consider one distinct set of patterns \( \mathcal{W}_k \) for each component \( k \). At each training round, each set \( \mathcal{W}_k \) is extended and pruned independently from the others. Note that all these automata are trained simultaneously using a common set of features. This process results in \( K \) automata, which are intersected on the fly\(^8\) using “lazy” composition. In our experiments, we also consider the case where we additionally combine the automaton representing complete tag sequences: this has the beneficial effect to restrict the combinations of subtags to values that actually exist in the data.

3.3 Pruned language models

Another approach for computing \( \mathcal{W} \) assumes that useful dependencies between tags can be identified using an auxiliary language model (LM) trained without paying any attention to observation sequences. A pattern \( w \) will then be deemed useful for the labelling task only if \( w \) is a useful history in a LM of tag sequences. This strategy was implemented by first training a compact \( p \)-gram LM with entropy pruning\(^9\) (Stolcke, 1998) and including all the surviving histories in \( \mathcal{W} \). In a second step, we train the complete CRF as usual, with all observation features and \( \ell_1 \) penalty to further prune the parameter set.

<table>
<thead>
<tr>
<th></th>
<th>cz</th>
<th>de</th>
</tr>
</thead>
<tbody>
<tr>
<td>train set</td>
<td>38,727</td>
<td>40,474</td>
</tr>
<tr>
<td>development set</td>
<td>5,228</td>
<td>5,000</td>
</tr>
<tr>
<td>test set</td>
<td>4,213</td>
<td>5,000</td>
</tr>
<tr>
<td># PoS</td>
<td>12</td>
<td>54</td>
</tr>
<tr>
<td># attributes</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td># full tags</td>
<td>1,924</td>
<td>781</td>
</tr>
</tbody>
</table>

Table 1: Corpus description

3.4 Maximum entropy language models

Another technique, which combines the two previous ideas, relies on Maximum Entropy LMs

---

\(^7\)cf. the discussion in (Vieira et al., 2016, § 4).

\(^8\)Formally, each \( A[\mathcal{W}_k] \) has transitions labelled with elements of \( \mathcal{Y}_k \); lazy intersection operates on “generalized” transitions, where each label \( z \) is replaced with \( [?, \ldots, z, \ldots, ?] \), where \( ? \) matches any symbol. \( A[\mathcal{W}] \) is the intersection \( \bigcap_k A[\mathcal{W}_k] \) and is labelled with completely specified tags.

\(^9\)Starting with a full back-off \( n \)-gram language model, this approach discards \( n \)-grams if their removal causes a sufficiently small drop in cross-entropy. We used the implementation of Stolcke (2002).
(MELMs) (Rosenfeld, 1996). MELMs decom-
pose the probability of a sequence $y_1 \ldots y_T$ using
the chain rule, where each term $p_A(y_t|y_{<t})$ is a lo-
cally normalized exponential model including all
possible ngram features up to order $p$:

$$p(y_t|y_{<t}; \lambda) = Z(\lambda)^{-1} \exp \lambda^T G(y_1 \ldots y_T)$$

In contrast to globally normalized models, the
complexity of training remains linear wrt. $|\mathcal{Y}|$, ir-
respective of $p$. It it also straightforward both to
(a) use a $\ell_1$ penalty to perform feature selection;
(b) include features that only test specific compo-
ents of a complex tag. For an order $p$ model, our
feature functions evaluate all $n$-grams (for $n \leq p$)
of complete tags or of one specific component:

$$G_w(y_1, \ldots, y_T) = \mathbb{I}(y_{t-n+1} \ldots y_t = w)$$

$$G_w(y_1, \ldots, y_T) = \mathbb{I}(z_{k,t-n+1} \ldots z_{k,t} = u)$$

Once a first round of feature selection has been
performed,\footnote{As the LM building step only look at labels, we tune
the regularization to optimize the perplexity of the LM on a
development set.} we compute $A[|W|]$ as explained
above. The last step of training reintroduces the
observations and estimates the CRF paramaters. A
variant of this approach adds extra gappy features to
the $n$-gram features. Gappy features at order $p$ test whether some label $u$ occurs in the remote
cell window around position $t$. These tests greatly
increase the feature count and are not provided for
all label patterns: for unigram patterns, we test the
presence of all unigrams and bigrams of words in
a window of 5 words; for bigrams patterns we only
test for all unigrams in a window of 3 words. Con-
textual features are not used for larger patterns.

4 Experiments

4.1 Training protocol

The following protocol is used throughout: (a)
identify $\mathcal{W}$ (§3) - note that this may imply to tune a
regularization parameter; (b) train a full model (in-
cluding tests on the observations for each pattern
in $\mathcal{W}$) using $\ell_1$ regularization and a very small $\ell_2$
term to stabilize convergence. The best regulariza-
ation in (a) and (b) is selected on development data
and targets either perplexity (for LMs) or label ac-
curacy (for CRFs).

4.2 Datasets and Features

Experiments are run on two MRLs: for Czech, we
use the CoNLL 2009 data set (Hajič et al., 2009)
and for German, the Tiger Treebank with the split
of Fraser et al. (2013)). Both datasets include rich
morphological attributes (cf. Table 1).

All the patterns in $\mathcal{W}$ are combined with lexical
features testing the current word $x_t$, its prefixes
and suffixes of length 1 to 4, its capitalization and
the presence of digit or punctuation symbols. Ad-
ditional contextual features also test words in a lo-
cal window around position $t$. These tests greatly
increase the feature count and are not provided for
all label patterns: for unigram patterns, we test the
presence of all unigrams and bigrams of words in
a window of 5 words; for bigrams patterns we only
test for all unigrams in a window of 3 words. Con-
textual features are not used for larger patterns.

4.3 Results

We consider several baselines: Maxent and
MEMM models, neither of which considers la-
bel dependencies in training, a linear chain CRF\footnote{Using
the implementation of Lavergne et al. (2010).}
and our own implementation of the group lasso of
Vieira et al. (2016). For the latter, we contrast two
setups: one where each pattern in $\mathcal{W}$ gives rise to
one single feature, and one where it is conjoined
with tests on the observation.\footnote{As suggested by the
authors themselves in fn 4.} All scores in Ta-
ble 2 are label accuracies on unseen test data.

As expected, Maxent and MEMM are outper-
formed by almost all variants of CRFs, and their
scores are only reported for completeness. Group
lasso results demonstrate the effectiveness of
using contextual information with high order fea-
tures: the gain is $\approx 0.7$ points for both languages
and all values of $p$. Greedy $\ell_1$ achieves accu-
rate results similar to group lasso, suggest-
ing that $\ell_1$ penalty alone is effective to select high-
order features. It also yields slightly smaller mod-
els and very comparable training time across the
board: indeed, greedy parameter selection strate-
gies imply multiple rounds of training which are
overall quite costly, due to the size of the full la-
bel set. Testing individual subtags (§ 3.2) results in
a slight improvement ($\approx+0.3$) in accuracy over
Greedy $\ell_1$. When using an additional automata
for the full tag, we get a larger gain of $\approx 0.6$ points
for Czech, slightly less for German: including a
model for complete tags also prevents to gener-
In this work, we have explored ways to take advantage of the flexibility offered by implementations of VoCRFs based on finite-state techniques. We have proposed strategies to include tests on sub-parts of complex tags, as well as to select useful label patterns with auxiliary unconditional LMs. Experiments with two MRLs with large tagsets yielded consistent improvements (∼+0.8 points) over strong baselines. They offer new perspectives to perform feature selection in high order CRFs.

In our future work, we intend to also explore how to complement ℓ₁ penalties with terms penalizing more explicitly the processing time; we also wish to study how these ideas can be used in combination with neural models.

Acknowledgements

The authors wish to thank the reviewers for their useful comments and suggestions. This work has been partly funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 645452 (QT21).
References


