Type-Supervised Hidden Markov Models for Part-of-Speech Tagging with Incomplete Tag Dictionaries

Dan Garrette
Department of Computer Science
The University of Texas at Austin
dhg@cs.utexas.edu

Jason Baldridge
Department of Linguistics
The University of Texas at Austin
jbaldrid@utexas.edu

Abstract

Past work on learning part-of-speech taggers from tag dictionaries and raw data has reported good results, but the assumptions made about those dictionaries are often unrealistic: due to historical precedents, they assume access to information about labels in the raw and test sets. Here, we demonstrate ways to learn hidden Markov model taggers from incomplete tag dictionaries. Taking the MIN-GREEDY algorithm (Ravi et al., 2010) as a starting point, we improve it with several intuitive heuristics. We also define a simple HMM emission initialization that takes advantage of the tag dictionary and raw data to capture both the openness of a given tag and its estimated prevalence in the raw data. Altogether, our augmentations produce improvements to performance over the original MIN-GREEDY algorithm for both English and Italian data.

1 Introduction

Learning accurate part-of-speech (POS) taggers based on plentiful labeled training material is generally considered a solved problem. The best taggers obtain accuracies of over 97% for English newswire text in the Penn Treebank, which can be considered as an upper-bound that matches human performance on the same task (Manning, 2011). However, as Manning notes, this story changes as soon as one is working with different assumptions and data, including having less training data, different kinds of training data, other languages, and other domains. Such POS tagging work has been plentiful and includes efforts to induce POS tags without labels (Christodoulopoulos et al., 2010); learn from POS-tag dictionaries (Ravi et al., 2010), incomplete dictionaries (Hasan and Ng, 2009) and human-constructed dictionaries (Goldberg et al., 2008); bootstrap taggers for a language based on knowledge about other languages (Das and Petrov, 2011), and creating supervised taggers for new, challenging domains such as Twitter (Gimpel et al., 2011).

Here, we focus on learning from tag dictionaries. This is often characterized as unsupervised or weakly supervised training. We adopt the terminology *type-supervised* training to distinguish it from unsupervised training from raw text and supervised training from word tokens labeled with their parts-of-speech. Work on type-supervision goes back to (Merialdo, 1994), who introduced the still standard procedure of using a bigram Hidden Markov Model (HMM) trained via Expectation Maximization.

Early research appeared to show that learning from types works nearly as well as learning from tokens, with researchers in the 1990s obtaining accuracies up to 96% on English (e.g. Kupiec (1992)). However, the tag dictionaries in these cases were obtained from labeled tokens. While replicating earlier experiments, Banko and Moore (2004) discovered that performance was highly dependent on cleaning tag dictionaries using statistics gleaned from the *tokens*. This greatly simplifies the job of a type-supervised HMM: it no longer must entertain entries for uncommon word-tag pairs (or mistaken pairs due to annotation errors), which otherwise stand on equal footing with the common ones. When the full, noisy tag dictionary was employed, Banko and Moore found accuracies dropped from 96% to 77%.

Banko and Moore's observations spurred a new line of research that sought to improve performance in the face of full, noisy dictionaries; see Ravi and

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 821–831, Jeju Island, Korea, 12–14 July 2012.

Knight (2009) for an overview. The highest accuracy achieved to date under these assumptions is 91.6% (Ravi et al., 2010). However, as is often noted (including by the authors themselves), many papers that work on learning taggers from tag dictionaries make unrealistic assumptions about the tag dictionaries they use as input (Toutanova and Johnson, 2008; Ravi and Knight, 2009; Hasan and Ng, 2009). For example, tag dictionaries are typically constructed with every token-tag pair in the data, including those that appear only in the test set. This means that the evaluation of these taggers does not measure how they perform on sentences that contain unseen words or unseen word-tag pairs, a likely occurrence in real use of a trained tagger.

We show that it is possible to achieve good tagging accuracy using a noisy and incomplete tag dictionary that has no access to the tags of the raw and test data and no access to the tag frequency information of the labeled training data from which the dictionary is drawn. We build on Ravi et al.'s (2010) model minimization approach, which reduces dictionary noise by greedily approximating the minimum set of tag bigrams needed to cover the raw data and exploits that information as a constraint on the initialization of the model before running EM. We extend their method in four distinct ways.

1. Enable the algorithm to be used with incomplete dictionaries by exploiting the type-based information provided by the tag dictionary and raw text to initialize EM, and by training a standard supervised HMM on the output of EM.
2. Improve the greedy procedure to find a better minimized set of tag-tag bigrams.
3. Modify the method to return only the set of bigrams required to tag sentences instead of keeping all bigrams chosen by minimization.
4. Exploit the paths found during minimization as a direct initialization for EM.

Together, these improvements make it possible to use model minimization in a realistic context, and obtain higher performance: on English, results go from 63.5% for a vanilla HMM to 82.1% for an HMM that uses strategies to deal with unknowns, then to 85.0% with Ravi and Knight's minimization and finally to 88.5% with our enhancements.

2 Supervision for HMMs

Hidden Markov Models (HMMs) are well-known generative probabilistic sequence models commonly used for POS-tagging. The probability of a tag sequence given a word sequence is determined from the product of emission and transition probabilities:

$$P(t|w) \propto \prod_{i=1}^{N} P(w_i|t_i) \cdot P(t_i|t_{i-1})$$

HMMs can be trained directly from labeled data by calculating maximum likelihood estimates or from incomplete data using Expectation Maximization (EM) (Dempster et al., 1977). We use both strategies in this work: EM is used to estimate models that can automatically label raw tokens, and then a new HMM is estimated from that auto-labeled data.

2.1 Token-supervised training

We use a simple but effective smoothing regime to account for unknown words and unseen tag-tag transitions. For emissions:

$$P(w_i|t_i) = \frac{C(t_i, w_i) + \alpha(t_i) P_{uni}(w_i)}{C(t_i) + \alpha(t_i)}$$

where $P_{uni}(w_i)$ is the unigram probability of w_i, and $\alpha(t_i)$ is a tag specific amount of mass for smoothing. We use one-count smoothing (Chen and Goodman, 1996), where $\alpha(t_i)$ is based on the number of words that occur with t_i once:

$$\alpha(t_i) = |w_i : C(t_i, w_i) = 1|$$

Since open-class tags occur more frequently with words that appear once, they will reserve more mass for unknown words than closed-class tags will. The transition distributions are smoothed in a similar fashion:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \lambda(t_{i-1}) P_{uni}(t_i)}{C(t_{i-1}) + \lambda(t_{i-1})}$$

$$\lambda(t_{i-1}) = |t_i : C(t_{i-1}, t_i) = 1|$$

This simple scheme is quite effective: an HMM trained on the Penn Treebank sections 0-18 and evaluated on sections 19-21 and smoothed in this way obtains 96.5% accuracy. We do not use gold standard labels elsewhere for this paper, but do use this model on the output of type-supervised HMMs.

2.2 Type-supervised training

We are primarily interested in learning taggers from tag dictionaries combined with unlabeled text. As is standard, we use EM to iteratively estimate the transition and emission probability parameters to maximize the likelihood of unlabeled data. It is known, however, that EM has particular problems learning a good HMM for POS tagging (Johnson, 2007; Ravi and Knight, 2009). One reason is that EM generally tries to learn probability distributions that are fairly uniform while POS tag frequencies are quite skewed. For example, "a" appears in the training data with seven different tags, but 99.9% of "a" tokens are determiners. Thus, the accuracy of anything approaching a uniform distribution for "a" tags will suffer greatly. In the context of unsupervised POS tagging models, modeling this distinction greatly improves results (Moon et al., 2010). Here, we can simply exploit the tag dictionary and raw data.

An initial set of parameters for the transitions and emissions must be supplied as the input to EM. Given just a tag dictionary, the simplest initialization is to set all tag transitions to be uniform, ranging over all tag continuations, while for emissions, a uniform distribution over all words that occur with the tag is assigned. This may be appropriate when a complete tag dictionary is available, including complete information for words that appear only in the test data. This is because there will never be any unknown words during model estimation or inference. Likewise, there will never be a situation where the tag dictionary rules out all possible tag transitions between two adjacent tokens in training or testing. As a result, no smoothing is needed in this scenario.

The problem with this is that estimating a model based on type-supervision requires raw text, and if we have an incomplete tag dictionary, some of the words in that text will be missing from the tag dictionary. In a Bayesian setting, priors provide mass for such tokens; models are estimated using either Gibbs sampling or variational inference (Johnson, 2007). However, we use vanilla EM here; as a consequence, once a parameter is zero, it is always zero. We thus need to ensure that mass is reserved for words outside the tag dictionary at the start of EM. (For transitions, uniform distributions are sufficient since the set of tags is closed.)

2.3 Emission probability initialization

The simplest way to initialize the emission distributions is to assign a count of one to every entry in the tag dictionary, and one count for unknowns. Then, during each iteration of EM, the expectation step is able to estimate new non-zero counts for all possible emissions encountered in the raw corpus. This basic strategy allows one to train an HMM with EM using only an incomplete tag dictionary and raw text. However, this basic approach for emission probabilities produces bad unknown-word probabilities. Specifically, if for each tag we simply assume one count for each entry in the tag dictionary and one count for unknowns and then normalize, the probability of an unknown word having a specific tag is inversely correlated with the number of word types associated with the tag in the tag dictionary. In other words, a tag that appears with a smaller number of distinct words will be seen by the HMM as being a better candidate tag for an unknown word. Unfortunately this is the opposite behavior we want since closed-class tags like *determiner* and *preposition* are bad candidates for tagging novel words.

For type-supervised training, we can do much better. Note that $C(w,t)$ comes in two varieties: w is either found in the tag dictionary (known word types), or it is not (unknown word types). We refer to the later as td-unknown: these are words that occur in the raw word sequence used for EM but which do not occur in the tag dictionary. These are thus different unknowns from words have not been observed in the dictionary or in the raw set but which may be encountered at test time. Computing the full $C(w,t)$ is necessary since we want $P(w|t)$ to cover known and td-unknown words. We must thus determine both $C_{known}(w,t)$ and $C_{unk_{td}}(w,t)$.

First, we focus on calculating $C_{known}(w,t)$. If a word w appears $C(w)$ times in the raw corpus, and is seen with $|TD(w)|$ tags in the tag dictionary, then assume for each t in $TD(w)$:

$$C_{known}(w,t) = C(w) \;/\; |TD(w)|$$

and $C_{known}(w,t) = 0$ for all other t. In other words, we split $C(w)$, the count of w tokens in the corpus, evenly among each of w's possible tags. This provides us with an estimate of the true $C(w,t)$ by approximating the portion of the counts of each word

type that may be associated with that tag. Note that while this will give us zeros for any words that don't appear in the raw corpus, this is not a problem because EM training is based only on that corpus.

Second, we look at td-unknown word types: those in the raw data that are *not* found in the tag dictionary. Given the value $P(unk_{td}|t)$ for the likelihood of an unknown word given a tag t, we can compute estimated counts $C_{unk_{td}}(w,t)$ for a td-unknown word w using

$$C_{unk_{td}}(w,t) = C(w) \cdot P(unk_{td}|t)$$

where $C(w)$, again, comes from the raw corpus. This has the effect of spreading $C(w)$, the count of tokens of that unknown word w, across all of the possible tags, with each tag receiving a proportion of the total count as determined by $P(unk_{td}|t)$.

The challenge, then, is to compute $P(unk_{td}|t)$. For this, we have two potential sources of knowledge, the tag dictionary and the raw token sequence, each telling us complementary information.

First, the tag dictionary tells us about the openness of a tag—the likelihood that an unseen word will have that label—based on our previously-discussed intuition that we are more likely to see a new word with a tag that is known to be associated with many words already. Thus, we can estimate $P_{td}(unk_{td}|t)$ by simply normalizing the $|TD(t)|$ values:

$$P_{td}(unk_{td}|t) = \frac{|TD(t)|^2}{\sum_{t' \in Tags} |TD(t')|^2}$$

We exaggerate the differences between tags by squaring the $|TD(t)|$ terms to draw an even larger distinction between open and closed class types.

Unfortunately, if we calculate an estimated word count directly from this using $C_{unk_{td}}(w,t) = C(w) \cdot P_{td}(unk_{td}|t)$, the $C_{unk_{td}}(w,t)$ values would be taken without any regard to the overall likelihood of tag t. Since $C_{known}(NN)$ is very high, $C_{unk_{td}}(NN)$ will seem very low by comparison. Likewise, since $C_{known}(RB)$ is much lower, $C_{unk_{td}}(RB)$ will seem very high by comparison.

$P(unk_{td}|t)$ must account for the overall likelihood of t so that the $C_{unk_{td}}(w,t)$ values will be scaled appropriately according to the overall likelihood of t. For this, we use our second knowledge source: the raw data. Based on the $C_{known}(w,t)$ values as given above, the raw data tells us about the overall expectation of a word having a particular tag. From this, we can estimate the tag distribution for known words: $C_{known}(t) = \sum_{w' \in V} C_{known}(w',t)$ and then normalize to get $P_{known}(t)$.

Finally, we need to combine $P_{td}(unk_{td}|t)$ and $P_{known}(t)$ into a single $P(unk_{td}|t)$ that accounts for both the openness of a tag and its overall prevalence. We would like this combination to use the high $P_{known}(NN)$ to boost $P(unk_{td}|NN)$ and the low $P_{known}(RB)$ to dampen $P(unk_{td}|RB)$. So, we compute and normalize:

$$P(unk_{td}|t) \propto \frac{|TD(t)|^2}{\sum_{t' \in Tags} |TD(t')|^2} \cdot P_{known}(t)$$

2.4 Auto-supervised post-EM smoothing

The initialization accounting for td-unknown words given above allows EM to be run on the raw token sequence, but it provides no probability for words that are truly unseen (in either the tag dictionary or the raw data). Consequently, any novel words in the test set will have zero emission probabilities, leading to extremely low unknown-word accuracies.

To overcome this problem, we perform a simple post-processing step after EM, which we refer to as **auto-supervised** training. We take the HMM trained by EM and use it to label the raw corpus. This gives us an automatically-labeled corpus that can be used for standard *supervised* training (without EM) to produce a new HMM. The effect of this post-processing step is to smooth the counts learned from EM onto any new words encountered during testing. This procedure significantly improves the ability of the HMM to label unknown words.

As a final note, it would of course be possible to use other models at this stage, such as a Conditional Random Field (Lafferty et al., 2001).

3 Enhancing MIN-GREEDY

As was discussed above, one of the major problems for type-supervised POS-tagger training with EM is a tag dictionary with low-frequency entries such as the word "a" being associated with the *foreign word* tag when nearly all of its instances are as a determiner. To avoid the need for manually pruning the tag dictionary, Ravi and Knight (2009)

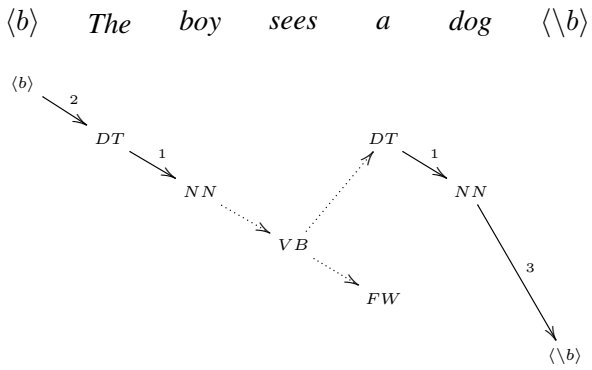

Figure 1: MIN-GREEDY graph showing a state in the first phase. Numbered, solid arrows: order of chosen bigrams; dotted: potential choices.

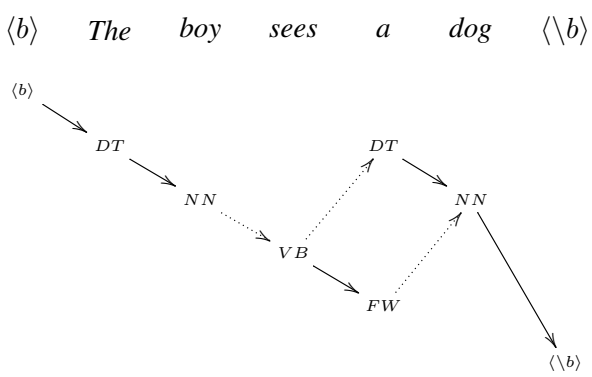

Figure 2: Start of the second MIN-GREEDY phase.

proposed that low-probability tags might be automatically filtered from the tag dictionary through a model minimization procedure applied to the raw text and constrained by the full tag dictionary. Ravi et al. (2010) develop a faster approach for model minimization using a greedy algorithm that they call MIN-GREEDY. It is this algorithm that we extend.

3.1 The original MIN-GREEDY algorithm

The MIN-GREEDY algorithm starts by initializing a graph with a vertex for each possible tag of each token in the raw data. The set of possible tags for each token is the set of tags associated with that word in the tag dictionary.Special *sentence start* and *sentence end* vertices are added to the graph for each sentence to mark its beginning and end. Unlike Ravi et al. (2010), we allow for an incomplete tag dictionary, meaning that our scenario has the additional complication that the tag set for some raw-corpus

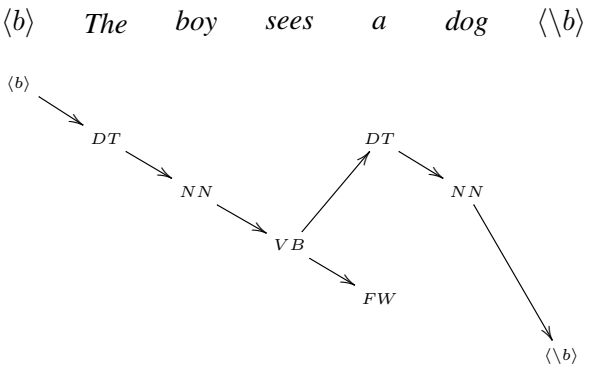

Figure 3: Potential MIN-GREEDY conclusion.

words will not be known. For these words, the full set of tags is used. Note that this increases the ambiguity and overall number of edges in the graph.

The MIN-GREEDY algorithm works in three phases: Greedy Set Cover, Greedy Path Completion, and Iterative Model-Fitting. In the first two phases, the algorithm chooses tag bigrams that form the edges of the graph. The goal of these phases is to select a set of edges that is sufficient to allow a path through every sentence in the raw corpus. The algorithm greedily selects these edges in an attempt to quickly approximate the minimal set of tag bigrams needed to accomplish this goal. In the final phase, the algorithm runs several iterations of EM in order to fit the bigram set to the raw data.

In the first phase, Greedy Set Cover, the algorithm selects tag bigrams in an effort to *cover* all of the word tokens. A word token is considered *covered* if there is at least one tag bigram edge connected to at least one of its vertices. At each iteration, the algorithm examines the entire graph, across all sentences, to find the tag bigram that, if added, would maximize the number of newly covered words.

Consider the graph in Figure 1. Assume, for the example, that this sentence comprises the entire raw corpus. At the start of the first phase, no tag bigrams are selected. On the first iteration, the algorithm chooses the tag bigram DT→NN because this tag bigram describes two edges for a total of four words newly covered: *The*, *boy*, *a*, and *dog*. On the second iteration, there are only three word tokens left uncovered: the start symbol, *sees*, and the end symbol. At this point, as the figure shows, there are five tag bigrams that would each result in covering one addi-

tional token. Since there are no tag bigrams whose choosing would result in covering more than one additional token, the algorithm randomly chooses one of these five. The algorithm iterates like this until all words are covered, as in, for example, Figure 2.

The second phase of the MIN-GREEDY algorithm, Greedy Path Completion, seeks to fill *holes* in the tag paths found in the graph. A *hole* is a potential edge that, if added, would connect two existing edges. At each iteration, the algorithm finds the tag bigram that, if selected, would maximize the number of holes that would be filled across all raw sentences.

The example graph in Figure 2 shows a potential start of the second phase. At this point, there are three tag bigrams that each fill one hole if selected, and the algorithm randomly selects one. Iteration continues until there is a complete tag path through each sentence in the raw corpus. One potential resolution for the example is given in Figure 3.

Once a set of tag bigrams has been generated that allows for a complete tag path through every sentence of the raw corpus, MIN-GREEDY begins its final phase: Iterative Model-Fitting. In this phase, the algorithm trains a succession of type-supervised HMM models. Each iteration trains an HMM and then uses it to tag the raw corpus, the result of which is used to prepare inputs for the next iteration.

Iterative Model-Fitting begins with the minimized set of bigrams returned from the second phase of MIN-GREEDY. This set is used as a hard constraint on the allowable tag bigrams during type-supervised HMM training. While EM is running, the only tag transitions that are counted are those that fall into the minimized tag bigram set; all other transition counts are ignored. Once an HMM has been trained, it is immediately used to tag the raw corpus, producing a set of auto-labeled sentences. For the second iteration of the phase, we extract a constrained tag dictionary from the auto-labeled corpus by simply taking every word/tag pair appearing in the data. This new tag dictionary is a subset of the original, full, tag dictionary, and hopefully has fewer low-frequency entries that would cause problems for EM.

We use this constrained tag dictionary to again perform type-supervised HMM training, but without any constraints on the allowable tag bigrams. This produces our third HMM. Using this HMM, we can, again, tag the raw corpus, producing another set of auto-labeled sentences. We can then extract the set of tag bigrams appearing in this data to produce a new set of tag transition constraints, similar to what was returned by the second phase. With this set of tag transition constraints, and the full tag dictionary, we can perform another round of type-supervised HMM training, and repeat the entire process.

The third MIN-GREEDY phase continues iterating, alternating between training an HMM using a constrained set of tag transitions and training one using a constrained tag dictionary. The size of the set of constrained tag bigrams produced is tracked on each iteration, and the algorithm is considered to have converged when this value changes by less than five percent. The final result of the MIN-GREEDY algorithm is a trained HMM.

The evaluation of the MIN-GREEDY algorithm, as described in Ravi et al. (2010), was performed only for scenarios with a complete tag dictionary (including all raw and test word types). As such, no techniques were described for handling unknown words. Because we are interested in the more realistic scenario of an incomplete tag dictionary, we augment the original MIN-GREEDY setup with the smoothing techniques described above.

3.2 Improving tag bigram selection

One of the major problems with the MIN-GREEDY algorithm is that its heuristics for choosing the next tag bigram frequently result in many-way ties. In the first two phases of MIN-GREEDY, the greedy procedure looks for the tag bigram that will have the most positive impact. In the Greedy Set Cover phase this means choosing the tag bigram that would cover the most new tokens, and in the Greedy Path Completion phase this means choosing the tag bigram that would fill the most holes. However, it is frequently the case that there are many distinct tag bigrams that would cover the most new tokens or fill the most holes, leaving the MIN-GREEDY algorithm with no choice but to randomly select from these options. Since there are frequently cases of having many dozens of options, it is clear that some of those choices *must* be better than others, even though MIN-GREEDY does not make a distinction and considers them all to be equally good choices.

Consider the example in Figure 1 representing a possible state of the minimization graph. To have

reached this stage, tag bigram DT→NN would have been chosen since it covered the highest number of tokens: four. Additionally, $\langle b\rangle$→DT and NN→$\langle\backslash b\rangle$ could have been chosen as the second and third tag bigrams since they tied for the most new tokens covered: one. For the state shown in this figure, there is only one uncovered token, *sees*, but three tag bigrams that cover it. Since each of these tag bigrams covers exactly one new word, they are all considered by MIN-GREEDY to be equally good choices as the next tag bigram for inclusion, and the algorithm will choose one at random. However, it should be clear that the VB→FW tag bigram is wrong while the other two would lead to a correct answer. As such, we would like for the algorithm to avoid choosing VB→FW, and to pick one of the others.

In order to push the algorithm into choosing the right tag bigrams in these otherwise ambiguous situations, we have added an additional criterion to the bigram-choosing heuristic: after narrowing down the set of tag bigrams to those that cover the most new tokens, we further narrow the choice of bigrams by minimizing the number of new word-type/tag pairs that would be added to the result. Consider our example. If we choose the tag bigram NN→VB or VB→DT, then exactly one new word-type/tag pair would be added to our result: *sees*/VB (since *boy*/NN and *a*/DT would already have been added by the incorporation of previous selected tag bigrams). By contrast if we choose the tag bigram VB→FW then *two* new word-type/tag pairs would be added: *sees*/VB and *a*/FW.

Minimizing the number of new word/tag pairs added by the algorithm has two main advantages. First, it keeps the selected bigrams focused on the same vertices, which results in fewer holes that the Greedy Path Completion phase must deal with. Secondly, it keeps the selected bigrams focused on more common tags for each word type, such as *a*/DT, and keeps it away from rare tags, such as *a*/FW.

3.3 Only tag bigrams on minimization paths

As was described above, the output of MIN-GREEDY's second stage is a minimized set of tag bigrams which is used as a constraint on the first iteration of the third stage, Iterative Model-Fitting. However, in order to determine when to stop adding new bigrams during the first two phases, the MIN-GREEDY algorithm must try to find complete tag paths through each sentence in the raw corpus, stopping once a tag path has been found for each one. While the algorithm is trying to select only the tag bigrams that are necessary for a complete tagging, it happens frequently that bigrams are selected that are not actually used on any tag path.

Consider the example shown in Figure 3. The graph has a complete path through the sentence, but also contains an extraneous edge, VB→FW, that is not used on the path. Assuming that this tag bigram is not used on the tag path of any other sentence, it can safely be removed from the resultant set to produce a smaller set of tag bigrams, getting us even closer to the minimized set that we desire.

To find the set of tag bigrams excluding these extraneous edges, we modify the MIN-GREEDY algorithm. During the first and second phases of the algorithm, we check all raw data sentences for a completed path after each tag bigram is selected. If a completed path is found for a sentence, we store that path immediately. Once a path is found for every sentence, we extract the set of bigrams used on these paths, and pass that set, instead of the full set of selected bigrams, to the third phase of the algorithm.

Note that it is important that we store the completed paths as soon as they are completed. Since sentences are completed at different stages, and more tag bigrams are selected after some of these sentences are complete, it is inevitable that some sentences will end up with multiple complete tag paths by the end of the second phase. However, we seek only the *first* such path. Tag bigrams are selected in order of their impact, so bigrams selected earlier are better and should be preferred. Considering again the example in Figure 3, based on the frequency of the tags, it is likely that, given the presence of other sentences in the raw corpus, the tag path including bigrams VB→DT and DT→NN would be found before the one including VB→FW and FW→NN. Since they are more frequent bigrams, we would want to keep the first path even if the second is completed at a later time.

The result of this improvement is a smaller, cleaner minimized tag bigram set to be delivered to the third phase of MIN-GREEDY.

Scenario	Total	Known	Unk.
0. Random baseline (choose tag randomly from tag dictionary)	63.53	65.49	2.38
1. HMM baseline (simple EM with tag dictionary and raw text)	69.20	71.42	0.27
2. HMM baseline + auto-supervised training	82.33	83.67	40.46
3. HMM baseline + auto-supervised training + emission initialization	82.05	83.27	44.31
4. MIN-GREEDY (Ravi et al., 2010) with add-one smoothing	74.79	77.17	0.45
5. MIN-GREEDY with add-one smoothing + auto-supervised	86.10	87.59	39.74
6. MIN-GREEDY with add-one smoothing + auto-supervised + emission init	85.02	86.33	44.28
7. 6 + enhanced tag bigram choice heuristic	86.71	88.08	43.93
8. 6 + restrict tag bigrams to tag paths of minimization-tagged output	87.01	88.40	43.74
9. 6 + HMM initialization from minimization-tagged output	**88.52**	**89.92**	**44.80**
10. 6 + 7 + 8 + 9	**88.51**	**89.92**	**44.80**

Table 1: English tagging accuracy using PTB sections 00-15 to build the tag dictionary. *Known* word types are those appearing in the tag dictionary.

3.4 EM initialization with minimization output

As a final improvement to MIN-GREEDY, we took the set of completed tag paths returned from the second phase of the algorithm, as described in the previous section, and used them as labeled data to initialize an HMM for EM training.

Since we modified MIN-GREEDY to produce a set of completed tag paths for sentences, we can take this to be a complete set of labels for the raw corpus. Furthermore, since we were careful about storing paths as soon as they become completed by the minimization process, and the tag bigrams are chosen in order of frequency, there will be more high-frequency bigrams than low-frequency. As a result, this labeling will contain good tag transitions and token labelings. As such, the labeled data produced by the second phase provides useful information beyond a simple set of sufficient bigrams: it contains legitimate frequency information that can be used to initialize the HMM. We, therefore, initialize an HMM directly from this data to start EM.

4 Evaluation[1]

English data. We evaluate on the Penn Treebank (Marcus et al., 1993). In all cases we use the first 47,996 tokens of section 16 as our raw data, sections 19–21 as our development set, and perform the final evaluation on sections 22–24.

[1]Source code, scripts, and data to reproduce the results presented here can be found at `github.com/dhgarrette/type-supervised-tagging-2012emnlp`

We evaluate two differently sized tag dictionaries. The first is extracted directly from sections 00–15 (751,059 tokens) and the second from sections 00–07 (379,908 tokens). The former contains 39,087 word types, 45,331 word/tag entries, a per-type ambiguity of 1.16 and yields a per-token ambiguity of 2.21 on the raw corpus (treating unknown words as having all 45 possible tags). The latter contains 26,652 word types, 30,662 word/tag entries, a per-type ambiguity of 1.15 and yields a per-token ambiguity of 2.03 on the raw corpus. In both cases, every word/tag pair found in the relevant sections was used in the tag dictionary: no pruning was performed.

Italian data. As a second evaluation, we use the TUT corpus (Bosco et al., 2000). To verify that our approach is language-independent without the need for specific tuning, we executed our tests on the Italian data without any trial runs, parameter modifications, or other changes. We divided the TUT data, taking the first half of each of the five sections as input to the tag dictionary, the next quarter as raw data, and the last quarter as test data. All together, the tag dictionary was constructed from 41,000 tokens consisting of 7,814 word types, 8,370 word/tag pairs, per-type ambiguity of 1.07 and a per-token ambiguity of 1.41 on the raw data. The raw data consisted of 18,574 tokens and the test contained 18,763 tokens.

Results We ran eleven experiments for each data set with results shown in Tables 1 and 2. All scores are reported as the percentage of tokens for which the correct tag was assigned. Accuracy is shown as

Scenario	PTB (00-07) Total	Known	Unk.	TUT Total	Known	Unk.
0. Random	64.98	68.04	2.81	62.81	76.10	1.58
1. HMM basic	69.32	72.70	0.56	60.70	73.77	0.51
2. HMM + auto-super	81.50	83.67	37.46	70.03	80.64	21.12
3. HMM + auto-super + init	81.71	83.62	42.89	70.89	80.91	24.74
4. MIN-GREEDY + add-1	68.86	72.20	0.92	53.96	65.49	0.84
5. MIN-GREEDY + add-1 + auto-super	80.78	82.88	38.02	70.85	82.41	17.60
6. MIN-GREEDY + add-1 + auto-super + init	80.92	82.80	42.64	71.52	81.56	**25.28**
7. 6 + enhanced bigram choice heuristic	86.69	88.83	43.07	71.48	81.57	24.98
8. 6 + restrict tag bigrams to tag paths	80.86	82.73	42.84	**72.86**	**83.45**	24.08
9. 6 + HMM init from minimization output	87.61	89.74	**44.18**	72.00	82.28	24.65
10. 6 + 7 + 8 + 9	**87.95**	**90.12**	43.74	71.99	82.50	23.57

Table 2: Tagging accuracy using PTB sections 00-07 and TUT to build the tag dictionary. *Known* word types are those appearing in the tag dictionary. Scenario numbers correspond to Table 1.

the Total (all word types), Known (word types found in the tag dictionary), and Unknown (word types not found in the tag dictionary).

Experiments 1–3 evaluate our smoothing techniques applied directly to the task of type-supervised HMM training with EM, without MIN-GREEDY. The basic HMM consistently beats the baseline random tagger, the auto-supervision technique makes an enormous improvement for both known and unknown words, and the the emission initialization yields a sizable improvement for unknown words.

Experiments 4–6 evaluated our reimplementation of MIN-GREEDY. We start with the most basic level of smoothing needed to work in a type-supervised scenario. For the smaller PTB tag dictionary and the TUT data, MIN-GREEDY actually has lower performance than the HMM alone. This indicates that if the tag dictionary has a low degree of ambiguity, then MIN-GREEDY can make the situation worse. However, with our smoothing techniques, we regain similar improvements as with the HMM.

Finally we performed experiments evaluating combinations of our improvements to MIN-GREEDY. Scenarios 7–9 show each improvement taken in turn. Scenario 10 shows the results for using all three improvements. For the English data, the best results are found when all the improvements are used. When taken individually, the bigram choice heuristic and HMM initialization from minimization output each consistently outperform the improved-MIN-GREEDY baseline on English. However, restricting the tag bigrams to that in the minimization-tagged output causes problems in the smaller PTB scenario, presumably falling to a local maximum like MIN-GREEDY that the other improvements are able to help the algorithm avoid.

Though the accuracy improvements are less than for English, the Italian results show that our MIN-GREEDY enhancements make an appreciable difference for a language and dataset for which the approaches considered were run sight unseen.

Error analysis One of the primary goals of model minimization is to automatically eliminate low-probability entries from the tag dictionary that might confuse the EM algorithm (Ravi et al., 2010). In order to see how well our techniques are able to identify and eliminate these unlikely word/tag pairs, we analyzed the tagging errors from each experiment. In doing so, we discovered that the two of the most problematic words for the EM algorithm are “a” and “in”. We ran further experiments explore what was happening with those words. The results, using PTB sections 00–07 are shown in Table 3.

In PTB sections 00-07 the word “a” appears 7630 times and with 7 different tags. This includes 7621 occurrences with tag DT, 3 with tag SYM (symbol), and 1 time with LS (list item marker). As such, we would want the HMM to lean heavily toward tag DT when tagging the token “a”. Unfortunately, the rare tags confuse the EM procedure and end up with dis-

tok	model output	tokens tagged by scenario					
		3	6	7	8	9	10
a	DT	32	4	4	4	2424	2425
	LS	1531	0	0	0	0	0
	SYM	731	2356	2305	2356	0	0
in	IN	12	15	2024	4	2042	2047
	FW	1922	1910	0	0	0	0
	RP	20	27	0	2037	0	0

Table 3: Number of times, for the words "a" and "in", the tagger trained by the particular scenario selected the given tag. Experiments used PTB sections 00-07 for the initial tag dictionary. Scenario numbers correspond to Table 1.

proportionately high probabilities. Our experiment training an HMM without minimization (scenario 3) resulted in 1531 "a" tokens being tagged LS, 731 as SYM, and only 32 tagged as DT.

The situation is similar with the word "in", which appears 6155 times with 5 different tags in the 8 sections. Of these, 6073 occurrences are tagged IN (preposition), 63 are RP (particle), and 1 is FW (foreign word). Again, EM without minimization is confused by the rare tokens, assigning FW 1922 times and IN 12 times.

The minimization procedure attempts to overcome this problem by removing unlikely tags from the tag dictionary automatically. As is show in Table 3, MIN-GREEDY without our enhancements is able to reject the problematic LS as a tag for "a", but unable to do so for SYM, resulting in 2356 tokens tagged SYM and only 4 tagged DT. Similarly, MIN-GREEDY is unable to reject FW as a tag for "in".

Our enhancements to MIN-GREEDY improve the situation. More careful choosing of bigrams during minimization results in the avoidance of LS and FW (but not SYM) for "a" as well as FW and RP for "in". Restricting the tag bigrams output from MIN-GREEDY to just those on tag paths avoids LS and FW for "a" and FW for "in". Finally, using the tagged sentences from MIN-GREEDY as noisy supervision for EM initialization eliminates all rare tags, as does the use of all three enhancements together.

5 Conclusion

Our results show it is possible to create accurate POS-taggers using type-supervision with incomplete tag dictionaries by extending the MIN-GREEDY algorithm of Ravi et al. (2010). The most useful change we made to the MIN-GREEDY procedure was the implementation of a better heuristic for picking tag bigrams. An intuitive and straightforward emission initialization provides the necessary basis to run EM on a given raw token sequence. Using EM output on this raw sequence as auto-labeled material to a supervised HMM then proves highly effective for generalization to new texts containing previously unseen word types.

Vaswani et al (2010) explore the use of minimum description length principles in a Bayesian model as a way of capturing model minimization, inspired by the MIN-GREEDY algorithm. The advantage there is that only a single objective function needs to be optimized, rather than having initialization followed by an iterative back and forth with pruning of tag-tag pairs. Our own next steps are to move in a similar direction to explore the possibilities for encoding the intuitions we developed for initialization and minimization as a single generative model.

Goldberg et al. (2008) note that fixing noisy dictionaries by hand is actually quite feasible, and suggest that effort should focus on exploiting human knowledge rather than just algorithmic improvements. We agree; however, our ultimate motivation is to use this work to tackle bootstrapping from very small tag dictionaries or dictionaries obtained from linguists or resources other than a corpus, and for tag sets that are more ambiguous (e.g., supertagging for CCGbank (Hockenmaier and Steedman, 2007)). Such efforts require automatic *expansion* of tag dictionaries, which then need be constrained based on available raw token sequences using methods such as those explored here. In this respect, the somewhat idiosyncratic noise in the corpus-derived dictionaries used here make a good test.

Acknowledgements

We thank Yoav Goldberg, Sujith Ravi, and the reviewers for their feedback. This work was supported by the U.S. Department of Defense through the U.S. Army Research Office (grant number W911NF-10-1-0533) and via a National Defense Science and Engineering Graduate Fellowship for the first author.

References

Michele Banko and Robert C. Moore. 2004. Part-of-speech tagging in context. In *Proceedings of COLING*, pages 556–561, Geneva, Switzerland.

Cristina Bosco, Vincenzo Lombardo, Daniela Vassallo, , and Leonardo Lesmo. 2000. Building a treebank for Italian: a data-driven annotation schema. In *Proceedings of LREC*.

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of ACL*, pages 310–318, Santa Cruz, California, USA.

Christos Christodoulopoulos, Sharon Goldwater, and Mark Steedman. 2010. Two decades of unsupervised pos induction: How far have we come? In *Proceedings of EMNLP*.

Dipanjan Das and Slav Petrov. 2011. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of ACL-HLT*, pages 600–609, Portland, Oregon, USA.

Arthur P. Dempster, Nan M. Laird, and Donald. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39:1–22.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of ACL-HLT*, pages 42–47, Portland, Oregon, USA.

Yoav Goldberg, Meni Adler, and Michael Elhadad. 2008. EM can find pretty good HMM POS-taggers (when given a good start). In *Proceedings ACL*, pages 746–754.

Kazi Saidul Hasan and Vincent Ng. 2009. Weakly supervised part-of-speech tagging for morphologically-rich, resource-scarce languages. In *Proceedings of EACL*, pages 363–371, Athens, Greece.

Julia Hockenmaier and Mark Steedman. 2007. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Mark Johnson. 2007. Why doesn't EM find good HMM POS-taggers? In *Proceedings EMNLP-CoNLL*, pages 296–305.

Julian Kupiec. 1992. Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6(3):225–242.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289. Morgan Kaufmann.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In Alexander Gelbukh, editor, *Proceedings of CICLing*, volume 6608 of *Lecture Notes in Computer Science*, pages 171–189.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Bernard Merialdo. 1994. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171.

Taesun Moon, Katrin Erk, and Jason Baldridge. 2010. Crouching dirichlet, hidden markov model: Unsupervised POS tagging with context local tag generation. In *Proceedings of EMNLP*, pages 196–206, Cambridge, MA.

Sujith Ravi and Kevin Knight. 2009. Minimized models for unsupervised part-of-speech tagging. In *Proceedings of ACL-AFNLP*.

Sujith Ravi, Ashish Vaswani, Kevin Knight, and David Chiang. 2010. Fast, greedy model minimization for unsupervised tagging. In *Proceedings of COLING*, pages 940–948.

Kristina Toutanova and Mark Johnson. 2008. A bayesian lda-based model for semi-supervised part-of-speech tagging. In *Proceedings of NIPS*.

Ashish Vaswani, Adam Pauls, and David Chiang. 2010. Efficient optimization of an mdl-inspired objective function for unsupervised part-of-speech tagging. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 209–214, Uppsala, Sweden.