ing the POS of a word. The question, then, is how to determine which suffixes are useful for the POS identification task in an *unsupervised* setting where we do not have any prior knowledge of language-specific grammatical constraints. This section proposes a method for identifying the "useful" suffixes and employing them to cluster the morphologically similar words. As we will see, our clustering algorithm not only produces soft clusters, but it also automatically determines the number of clusters for a particular language.

Before we describe how to identify the useful suffixes, we need to (1) induce all of the suffixes and (2) morphologically segment the words in our vocabulary.[2] However, neither of these tasks is simple for a truly resource-scarce language for which we do not have a dictionary or a knowledge-based morphological analyzer. As mentioned in the introduction, our proposed solution to both tasks is to use an unsupervised morphological analyzer that can be built just from an unannotated corpus. In particular, we have implemented an unsupervised morphological analyzer that outperforms Goldsmith's (2001) Linguistica and Creutz and Lagus's (2005) Morfessor for our English and Bengali datasets and compares favorably to the best-performing morphological parsers in MorphoChallenge 2005[3] (see Dasgupta and Ng (2007)).

Given the segmentation of each word and the most frequent 30 suffixes[4] provided by our morphological analyzer, our clustering algorithm operates by (1) clustering the similar suffixes and then (2) assigning words to each cluster based on the suffixes a word combines with. To cluster similar suffixes, we need to define the similarity between two suffixes. Informally, we say that two suffixes x and y are similar if a word that combines with x also combines with y and vice versa. In practice, we will rarely posit two suffixes as similar under this definition unless we assume access to a complete vocabulary – an assumption that is especially unrealistic for resource-scarce languages. As a result, we relax this definition and consider two suffixes x and y similar if $P(x \mid y) > t$ and $P(y \mid x) > t$, where $P(x \mid y)$ is the probability of a word combining with suffix x given that it combines with suffix y, and t is a threshold that we set to 0.4 in all of our experiments. Note that both probabilities can be estimated from an unannotated corpus.[5] Given this definition of similarity, we can cluster the similar suffixes using the following steps:

Creating the initial clusters. First, we create a *suffix graph*, in which we have (1) one node for each of the 30 suffixes, and (2) a directed edge from suffix x to suffix y if $P(y \mid x) > 0.4$. We then identify the strongly connected components of this graph using depth-first search. These strongly connected components define our initial partitioning of the 30 suffixes. We denote the suffixes assigned to a cluster the *primary keys* of the cluster.

Improving the initial clusters. Recall that we ultimately want to cluster the words by assigning each word w to the cluster in which w combines with all of its primary keys. Given this goal, it is conceivable that singleton clusters are not desirable. For instance, a cluster that has "s" as its only primary key is not useful, because although a lot of words combine with "s", they do not necessarily have the same POS. As a result, we improve each initial cluster by adding more suffixes to the cluster, in hopes of improving the resulting clustering of the words by placing additional constraints on each cluster. More specifically, we add a suffix y to a cluster c if, for each primary key x of c, $P(y \mid x) > 0.4$. If this condition is satisfied, then y becomes a *secondary key* of c. For each initial cluster c', we perform this check using each of the suffixes x' not in c' to see if x' can be added to c'. If, after this expansion step, we still have a cluster c* defined by a single primary key x that also serves as a secondary key in other clusters, then x is *probably ambiguous* (i.e., x can probably attach to words belonging to different POSs); and consequently, we remove c*. We denote the resulting set of clusters by C.

Populating the clusters with words. Next, for each word w in our vocabulary, we check whether w can be assigned to any of the clusters in C. Specifically, we assign w to a cluster c if w can combine with each of its primary keys and at least half of its secondary keys.

Labeling and merging the clusters. After populating each cluster with words, we manually label

[2] A vocabulary is simply a set of (distinct) words extracted from an unannotated corpus. We extracted our English and Bengali vocabulary from WSJ and Prothom Alo, respectively.

[3] http://www.cis.hut.fi/morphochallenge2005/

[4] We found that 30 suffixes are sufficient to cluster the words.

[5] For instance, we compute $P(x \mid y)$ as the ratio of the number of distinct words that combines with both x and y to the number of distinct words that combine with y only.