Digital Assignment 2

Kartikay Kaul

16BCE1057

Natural Language Processing

Professor R. Sridhar

**PART 1**

# Rule Based Tagging

A rule-based tagging approach uses contextual information to assign tags to unknown or ambiguous words. Disambiguation is done by analyzing the linguistic features of the word, its preceding word, its following word, and other aspects. Rule based taggers depend on dictionary or lexicon to get possible tags for the words to be tagged. Hand-written rules are used to identify the correct tag when a word has more than one possible tag. For example, if the preceding word is a determiner then the word in question must be noun. This information is coded in the form of rules. The rules may be context-pattern rules or as regexes compiled into an FSA that are intersected with lexically ambiguous sentence representations. TAGGIT, the first large rule-based tagger, used context-pattern rules. TAGGIT used a set of 71 tags and 3300 disambiguation rules. These rules disambiguated 77% of words in the million-word Brown University corpus.

Rule based taggers are knowledge-drive and rules are built manually. The rules are finite in amount (~ 1000). The LM and smoothing are explicitly defined in rule-based tagging.

A context frame rule might say something such as “If an ambiguous/unknown word X is preceded by a Determiner and followed by Noun, tag it as an Adjective.” On the other hand, the transformation based approached use a pre-defined set of handcrafted rules as well as automatically induced rules that are generated during training.

Morphology is a linguistic term which means how words are built up from smaller units of meaning known as morphemes. In addition to contextual, morphological information is also used by some models to aid in the disambiguation process. One such rule can be “If 5 an ambiguous or unknown word ends in a suffix ‘-ing’ and is preceded by a verb, label it a ‘Verb’”.

Some models also use information about capitalization and punctuation, the usefulness of which are largely dependent on the language being tagged.

In general, this tagging model usually requires supervised training i.e. a pre-annotated corpus. But in recent times, good amount of work has been done to automatically induce the transformation rules. One approach to automatic rule induction is to run an untagged text through a tagging model and get the initial output. A human then goes through the output of this first phase and corrects any erroneously tagged words by hand. This tagged text is then submitted to the tagger, which learns correction rules by comparing the two sets of data. Several iterations of this process are necessary before the tagging model can achieve considerable performance.

A case study done on a research paper shows a tagger that works automatically by recognizing its weaknesses, thereby incrementally improving its performance. The tagger initially tags by assigning each word its most likely tag, estimated by examining a large tagged corpus, without **regard to context**. In both sentences below, *run* would be tagged as a verb:

“The **run** lasted thirty minutes”

“We **run** three miles every day.”

The tagger would tend to tag words ending with “-*ous*” with adjective. This information is automatically derived from the corpus. The tagger makes use of the Brown corpus.

**Negatives**: Sometimes the rules are giving wrong meaning to the sentence. Rules are not context-dependent. High development cost. Not transportable. Time cost of tagging is high.

**Positives:** Freely available. Works Fairly Well. Supervised learning needs a pre-annotated corpus for training. There is high precision. The rules are linguistically motivated.

# Stochastic Tagging

The term can refer to any number of different approaches to the problem of POS tagging. Any model which somehow incorporates frequency or probability (using statistics) may be properly labelled stochastic.

The simplest stochastic taggers disambiguate words based solely on the probability that a word occurs with a tag. In other words, the tag encountered most frequently in the training set is the one assigned to an ambiguous instance of that word. The problem with this approach is that while it may yield a valid tag for a given word, it can also yield inadmissible sequence of tags.

An alternative to the word frequency approach is to calculate the probability of a given sequence of tags occurring. This is sometimes referred to as the *n-gram* approach,  referring to the fact that the best tag for a given word is determined by the probability that it occurs with the n previous tags. The most common algorithm for implementing an n-gram approach is known as the *Viterbi Algorithm*, a search algorithm which avoids the polynomial expansion of a breadth first search by "trimming" the search tree at each level using the best N *MLE*(where n represents the number of tags of the following word).

The next level of complexity that can be introduced into a stochastic tagger combines the previous two approaches, using both tag sequence probabilities and word frequency measurements. This is known as a Hidden Markov Model. The assumptions underlying this model are the following:

Each hidden tag state produces a word in the sentence. Each word is:

1. Uncorrelated with all the other words and their tags
2. Probabilistic depending on the N previous tags only

Hidden Markov Model taggers and visible Markov Model taggers may be implemented using the Viterbi algorithm, and are among the most efficient of the tagging methods discussed here. HMM's cannot, however, be used in an automated tagging schema, since they rely critically upon the calculation of statistics on output sequences (tag-states). The solution to the problem of being unable to automatically train HMMs is to employ the *Baum-Welch Algorithm*, also known as the *Forward-Backward Algorithm*. This algorithm uses word rather than tag information to iteratively construct a sequence to improve the probability of the training data.

If one takes a stochastic approach to POS tagging, it is necessary to make all of the necessary measurements and calculations to determine the values for the n-gram based transitional frequency values.

In order to create a matrix of transitional probabilities, it is necessary to begin with a tagged corpus upon which to base the estimates of those probabilities. For the purpose of exposition, we will consider how to go about determining those values using a bigram model, i.e. we will base our estimates on the immediate context of a words and will not consider any context further than one word away.

The first step in this process is to determine the probability. of each category's occurrence. This is a matter of straightforward probability calculation. In order to determine the probability of a noun occurring in each corpus, we divide the total number of nouns by the total number of words. So, if we had a hundred-word corpus and 20 of those words were nouns, the estimated probability of a noun occurring would be 20/100 or. 20. Likewise, if there were thirty determiners in the same corpus, the probability of a determiner occurring would be .30.

Next, we face the problem of determining transitional probabilities for sequences of words, which boils down to calculating the number of times that the event occurs given the occurrence of another event. This is known as conditional probability, for which the generic formula is:

*this formula can be read as “the probability of an event e occurring given the occurrence of another event e' is equal to the probability of both events occurring at once divided by the probability of the occurrence of e'.”*

We can put this formula to work for us in determining the transitional probabilities in the following way: Suppose that we wish to determine the probability of a noun following a determiner. We plug these categories into the probability formula:

In practice, a variation of this formula is used wherein category frequencies are used rather than category probabilities. So, the formula turns out to be:

The number of probabilities for a corpus equals the number of tags in the tag set squared. The start and end categories are always included in the calculation of N squared.

There is, however, a flaw in the formula for conditional probabilities, at least insofar as using it to determine transitional probabilities is concerned. The trouble is that words which occur with great frequency, such as nouns, get favoured too heavily during the disambiguation process, resulting in a decrease in the precision of the system. The problem is that the frequency of the category at i-1 was never considered. The solution is to slightly modify that equation to include the frequency of the context word:

here the denominator is the product of the frequencies of the words in the bigram, rather than just the frequency of the context word.

The major **disadvantage** of stochastic taggers is that they are computationally expensive given all the factors one must consider as shown in above formulations. The **advantage** is that it is more accurate mathematically than rule-based tagging. It is context dependent.

# Transformational tagging

TBL is a rule-based algorithm for automatic tagging of POS to a given text. TBL (transformation-based learning) transforms one state to another using transformation rules in order to find the suitable tag for each word. TBL allows one to have linguistic knowledge in a readable form. It extracts linguistic information automatically from corpora. The outcome of TBL is an ordered sequence of transformations of the form as shown below.

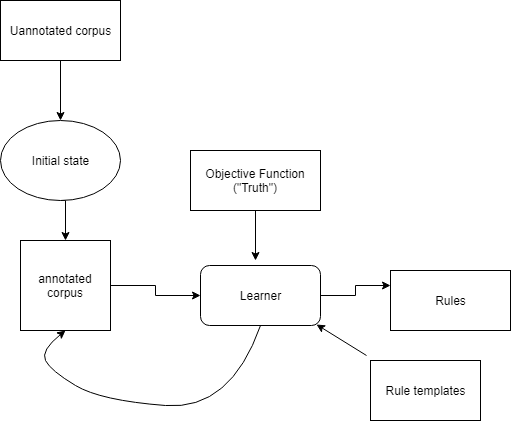
A typical transformation-based learner has an initial state annotator, a set of transformations and an objective function. usually starts with some simple solution to the problem. Then it runs through cycles. At each cycle, the transformation which gives more benefit is chosen and applied to the problem. The algorithm stops when the selected transformations do not add more value or there are no more transformations to be selected. This is like painting a wall with background color first, then paint different color in each block as per its shape or so. TBL is best suitable for classification tasks.

In TBL, accuracy is generally considered as the objective function. So, in each training cycle, the tagger finds the transformations that greatly reduce the errors in the training set. This transformation is then added to the transformation list and applied to the training corpus. At the end of the training, the tagger is run by first tagging the fresh text with initial-state annotator, then applying each transformation in order wherever it can apply.

A TBL consists of two phases:

* Training phase (usually iterated once): rules are learnt in this level
* Application phase (usually iterated many times): the rules are applied in the order they were learnt

Flow of training phase:



POS Tagging with TBL

1. Initial state

* Known words are tagged with their most frequent tag
* Unknown words are tagged with the most frequent tag in the training corpus, depending on the first letter of the word in question

1. Lexical tagging: the unknown words are tagged in isolation, based on their morphology and their immediate neighbour
2. Contextual tagging: All words are tagged in context

**DISADVANTAGES**

* Free word order thereby aggravating the contextual tagging
* Non-normalised spelling, resulting in fewer occurrences of identical word forms, i.e. a high ratio of word forms per lemma
* Training corpus is too small
* TBL does not provide tag probabilities.
* Training time is often intolerably long, especially on the large corpora which are very common in Natural Language Processing.

**ADVANTAGES**

* Small set of simple rules that are enough for tagging is learned.
* As the learned rules are easy to understand development and debugging are made easier.
* Interlacing of machine-learned and human-generated rules reduce the complexity in tagging.
* Transformation list can be compiled into finite-state machine resulting in a very fast tagger. A TBL tagger can be even ten times faster than the fastest Markov-model tagger
* TBL is less rigid in what cues it uses to disambiguate a word. Still it can choose appropriate cues.

PART B

