**CMPT 825: Natural Language Processing** **Spring 2008**

# Lecture 12 — Feb 4, 2008

*Lecturer: Anoop Sarkar* *Scribe: Milan Tofiloski*

### 12.0.1 Definitions

**Tagger** Program that tags a word ($w_i$) in a text with its part of speech (POS) tag $t_i$.

**Named Entity Task** The named entity task is to identify all named locations, named persons, named organizations, dates, times, monetary amounts, and percentages in text.

**Precision** Precision is the number of correct responses out of the total number of responses.

**Recall** Recall is the number of correct responses out of the number correct in the key.

**Key File** A key file is an annotated file containing the correct answers.

**F Measure**

$$\text{F-Measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{(\text{Recall} + \text{Precision})}$$

**Bayes' Rule**

$$\Pr(\text{Name-Classes}|\text{Word Sequence}) = \frac{\Pr(\text{Word Sequence}, \text{Name-Classes})}{\Pr(\text{Word Sequence})}$$

**Word Features** Features of a word that are used in characterizing the type of entity it is (i.e. whether a word is capitalized, whether a word contains digits or special punctuation, whether all letters are capitalized, etc.).

**Semantic Class** Defined as a list of words each having a common semantic feature.

**Information Extraction** Information extraction structures information from unstructured texts.

**Field Segmentation** Extracting information into a field structure. For example, possible fields to be extracted from a classified advertisement document would be date, item, and price. Another example would be where businesses segment resumes into name, address, position applied for, work history, etc. fields.

**Function Words** Commas, periods, semicolons, etc. are considered function words.

**kNN Word Similarity Method** Method for finding similar words, as described in [5]. Determines nearest neighbors based on words with related suffixes suffix-related words and the context of nearby words.

## 12.1 An Algorithm That Learns What's in a Name[1]

### 12.1.1 Overview

The named entity recognition problem is to identify all locations, organizations, persons, dates, times, monetary amounts, and percentages in a text. If one were to use handwritten rules, significant modifications are required when moving from one text source to another. The idea proposed in this paper uses context to determine whether a word(s) is a named entity.

Tag sequences of a document consist of a long series of consecutive NOT-A-NAME tags followed by few (mostly single) NAME tags. This differs from POS tagging where we are switching tags very frequently. Hence, name recognition is a classification problem (a word is classified as either NAME or NOT-A-NAME).

A traditional HMM model would have a small number of states within each name-class, each having some semantic significance (e.g. three states in the PERSON name-class would represent a first, middle and last name) where each of these three states would have some probability associated with emitting any word from the vocabulary. A bigram language model was chosen since it works well in practice.

A generative model where the HMM generates the sequence of words and labels is used. Bayes Rule is thus:

$$\Pr(\mathrm{NC}|\mathrm{W}) = \frac{\Pr(\mathrm{W},\mathrm{NC})}{\Pr(\mathrm{W})} \tag{12.1}$$

where NC stands for Name-Classes, and W stands for Word Sequence. The generation of words and name-classes is as follows:

1. Select a name-class, conditioning on the previous name-class and the previous word.

2. Generate the first word inside that name-class, conditioning on the current and previous name-classes.

3. Generate all subsequent words inside the current name-class, where each subsequent word is conditioned on its immediate predecessor (as per a standard bigram language model).

These three steps are repeated until the entire observed word sequence is generated. Using the Viterbi algorithm, the entire space of all possible name-class assignments is searched (maximizing the numerator $\Pr(\mathrm{W},\mathrm{NC})$).

The 'word-feature' of a word is a language-dependent characterization. For example, in Roman languages capitalization gives strong evidence of a name. Some word-features are: whether a word is capitalized, whether a word contains digits or special punctuation, whether all letters are capitalized, etc. The word-features aid in distinguishing and annotating monetary amounts, percentages, times and dates.

Let $c()$ represent the number of times the events occurred in the training data (the count). The probabilities are computed as:

$$\Pr(NC|NC_{-1}) = \frac{c(NC, NC_{-1}, w_{-1})}{c(NC_{-1}, w_{-1})} \tag{12.2}$$

$$\Pr(< w, f >_{first} |NC, NC_{-1}) = \frac{c(< w, f >_{first}, NC, NC_{-1})}{c(NC, NC_{-1}))} \tag{12.3}$$

$$\Pr(< w, f > | < w, f >_{-1}, NC) = \frac{c(< w, f >, < w, f >_{-1}, NC)}{c(< w, f >_{-1}, NC)} \tag{12.4}$$

### 12.1.2 Handling Of Unseen Words In The Training Corpus

All words that were not in the training data are labelled as belonging to the class ‘unknown’ (UNK). There are three ways an unknown word can appear in a bigram: as the current word, as the previous word, or as both. All unknown words are treated as a single token, UNK.

### 12.1.3 Back-Off Model

To compute the back-off weight $\lambda$ for

$$\Pr(\text{PERSON}|\text{NOT-A-NAME, ‘Mr.’})$$

where $c(\text{NOT-A-NAME, ‘Mr.’}) \approx c(\text{NOT-A-NAME})$, the estimate of

$$\Pr(\text{PERSON}|\text{NOT-A-NAME})$$

should be removed from the smoothing of the estimate of

$$\Pr(\text{PERSON}|\text{NOT-A-NAME, ‘Mr.’})$$

since the two have equal training, but the latter is a superior model since it conditions on more context.

For example, if the bigram ‘come hither’ was observed once in training and ‘come here’ was observed three times, while the word ‘come’ was not observed in the NOT-A-NAME class, then when computing Pr(‘hither’|‘come’, NOT-A-NAME), the algorithm would back off to the unigram probability Pr(‘hither’|NOT-A-NAME) with weight $\frac{1}{3}$ since the number of unique outcomes for the word-state for ‘come’ would be two, and the total number of times ‘come’ had been the preceding word in a bigram would be four (the bigram probability weight of $1/(1+\frac{2}{4}) = \frac{2}{3}$), and the back-off model weight of $1 - \frac{2}{3} = \frac{1}{3}$).

### 12.1.4 Examples

#### Example 1

Suppose we have the sentence ‘Mr. Jones eats.’

The correct annotation for this sentence is: ‘Mr. <ENAMEX TYPE =PERSON> Jones < /ENAMEX> eats.’ Here ‘Mr.’ is a good indicator of the next word beginning the PERSON name-class.

'Jones' is in the PERSON name-class and the other tokens are in the NOT-A-NAME class. The following likelihood is assigned to the sentence above:
Pr(NOT-A-NAME | START-OF-SENTENCE, +end+)
· Pr('Mr.' | NOT-A-NAME, START-OF-SENTENCE)
· Pr(+end+ | 'Mr.', NOT-A-NAME)
· Pr(PERSON | NOT-A-NAME, 'Mr.')
· Pr('Jones' | PERSON, NOT-A-NAME)
· Pr(+end+ | 'Jones', PERSON)
· Pr(NOT-A-NAME | PERSON, 'Jones')
· Pr('eats' | NOT-A-NAME, PERSON)
· Pr('.' | 'eats', NOT-A-NAME)
· Pr(+end+ | '.', NOT-A-NAME)
· Pr(END-OF-SENTENCE | NOT-A-NAME, '.')

**Example 2**

Let our annotated example sentence be: 'The Turkish company, <ENAMEX TYPE='LOCATION'> Birgen Air < /ENAMEX>, was using the plane to fill a charter commitment to a German company, ...'

Birgen Air has been labeled a LOCATION, when it is actually an ORGANIZATION (and is so marked in the key file). According to the tokenization rules, punctuation marks such as commas are treated as separate tokens, meaning that the 'word' directly preceding Birgen is ',' and the same 'word' directly follows 'Air'. The paper's IdentiFinder program is incapable of using the slightly wider context of a trigram that would include the word 'company' when predicting the beginning of this ORGANIZATION. Since 'Birgen' is an unknown word and the capitalized word 'Air' happens to have a very high unigram probability for appearing within a LOCATION (due to many training examples of 'Edwards Air Force Base'). For the same reason, 'UNK Air' is a very likely bigram in a LOCATION in the unknown word model. By using more context in the model, this error could have been prevented. But the downside of using more context, such as trigrams, is that in practice exponentially more training data is required.

It is actually detrimental to remove tokens such as commas out of the data stream and then reinsert them after name-finding is complete. Punctuation tokens, commas in particular, give great evidence of names, especially for LOCATIONs having the form 'City, State'. For example, in 'Los Angeles, California', the comma helps signal the end of the first LOCATION 'Los Angeles' and the beginning of the next LOCATION 'California'.

### 12.1.5 Results

The F-measure score was used to compute the quality of analysis. A response is considered half correct if both type and attribute are correct but only one boundary is correct. Also, a response is half-correct if both boundaries are correct along with the type are correct but not the attribute.

The formalisms or techniques presented in this paper are not new, but applying an HMM to this task (and the model itself) are novel. By using a fairly simple probabilistic model, finding names and other numerical entities can be performed with 'near-human performance' (an F-measure of 95 or above). The system in this paper performs as good, if not better, than state-of-the-art systems.

## 12.2 Unsupervised Learning of Field Segmentation Models for Information Extraction [2]

### 12.2.1 Overview

Information extraction systems are not generalized tools and hence require retraining for each different application. This paper looks at unsupervised learning of field segmentation models. The domains of interest are bibliographic citations and classified advertisements for apartment rentals.

General unconstrained HMMs (using EM) fail to detect useful field structure in either of the two domains of interest. But with small amounts of prior knowledge, the learning model can be significantly improved.

The general approach is:

- An HMM is fit to have the same number of hidden states as 'GOLD LABELS'????? using EM.
- Hidden state expectations are computed using the Forward-Backward algorithm.

- Emission models are initialized to near-uniform probability distributions with a small amount of added noise to break initial symmetry.

Smoothing techniques were not extensively investigated in this paper.

### 12.2.2 Diagonal Transition Model

For learning larger-scale patterns, the parametric form of the transition model is constrained, becoming:

$$\mathrm{P}(s_t|s_{t1}) = \begin{cases} \sigma + \frac{1-\sigma}{|S|} & \text{if } s_t = s_{t-1} \\ \frac{1-\sigma}{|S|} & \text{otherwise} \end{cases} \tag{12.5}$$

where $|S|$ is the number of states, and $\sigma$ is a global free parameter specifying the probability of a state transitioning to itself. This constraint improves performance: with 400 unannotated training documents the accuracy jumps from 48.8% to 70.0% for advertisements and from 49.7% to 66.3% for citations.

### 12.2.3 Hierarchical Mixture Emission Model

Each state also emits punctuation and English function words in addition to content words. When labeling decisions are made on the basis of the function words rather than the content words, this can become problematic.

A way of making certain words unavailable to the model is to emit those words from all states with equal probability. This can be accomplished with the following hierarchical mixture emission model

$$\mathrm{P}_h(w|s) = \alpha \mathrm{P}_c(w) + (1-\alpha)\mathrm{P}(w|s) \tag{12.6}$$

where $\mathrm{P}_c$ is the common word distribution, and $\alpha$ is a new global free parameter. Before a state emits a token it flips a coin, and with probability $\alpha$ it allows the common word distribution to generate the token, and the token is generated with probability $(1-\alpha)$ from its state-specific emission model.

The emission model of $\mathrm{P}_c$ was initialized to a list of stopwords without any reestimation. This improved average accuracy from 70.0% to 70.9%. The system learned the emission model of $\mathrm{P}_c$ using EM reestimation.

### 12.2.4 Boundary Model

Another source of error concerns field boundaries. Fields are more or less correct, but the boundaries are off by a few tokens even when punctuation or syntax make it clear to a human reader where the exact boundary should be. The paper models the fact that data fields often end with one of the few boundary tokens (eg. punctuation and new lines) which are shared across states.

The transition function for non-final states becomes:

$$\mathrm{P}(s|s^-) = \begin{cases} (1-\mu)(\lambda + \frac{1-\lambda}{|S^-|}), & \text{if } s = s^- \\ \mu(\lambda + \frac{1-\lambda}{|S^-|}), & \text{if } s = s^+ \\ \frac{1-\lambda}{|S^-|}, & \text{if } s \in S^- \setminus s^- \\ 0, & \text{otherwise} \end{cases} \tag{12.7}$$

where $s^- \in S^-$ is a non-final state, $s^+ \in S^+$ is a final state, $\lambda$ is the probability of staying within the field, and $\mu$ is the probability of transitioning to the final state given we are staying in the field.

The transition function for final states becomes:

$$\mathrm{P}(s|s^+) = \begin{cases} \sigma + \frac{1-\sigma}{|S|} & \text{if } s = s^- \\ \frac{1-\sigma}{|S|} & \text{if } s \in S^- \setminus s^- \\ 0 & \text{otherwise} \end{cases} \tag{12.8}$$

A test using both methods (supplying the boundary token distributions and learning them with reestimation during EM) was performed on both domains. With the advertisements data, learning the boundary emission model gives an increase from 70.0% to 70.4%, while specifying the list of allowed boundary tokens gives an increase to 71.9%. When combined with the given hierarchical emission model, accuracy rises to 72.7%. This is the best unsupervised result on the advertisements data with 400 training examples. With the citations data it was found that learning boundary emission model hurts accuracy, but being given the set of boundary tokens it improves accuracy from 66.3% to 68.2%.

### 12.2.5 Results

The bibliographic citations task should intuitively have greater success than the apartment rental classified ad task, but the results in this paper show otherwise.

**Baselines Used For Comparison**

One baseline is the most-frequent-field accuracy model, which labels all tokens with the same single label and is then mapped to the most frequent eld. This gives an accuracy of 46.4% on the advertisements data and 27.9% on the citations data.

Another baseline method is to presegment the unlabeled data using a heuristic based on punctuation, and then cluster the resulting segments using a Naive Bayes mixture model with the EM algorithm. This approach achieves an accuracy of 62.4% on the advertisements data and 46.5% on the citations data.

The last baseline trains a supervised first-order HMM from the annotated training data using maximum likelihood estimation. With 100 training examples, supervised models achieve an accuracy of 74.4% on the advertisements data, and 72.5% on the citations data. With 300 examples, supervised methods achieve accuracies of 80.4% on the citations data.

In the unconstrained approach, with 400 unannotated training documents, the accuracy is just 48.8% for the advertisements and 49.7% for the citations. This is better than the single state baseline but nowhere near as good as the supervised baseline.

With a semi-supervised approach, adding 5 annotated citations yields no improvement in performance, but adding 20 annotated citations to 300 unannotated citations boosts performance from 65.2% to 71.3%.

In the two different domains of classified advertisements and bibliographic citations, constraining the model class allows to restrict the search space of EM to models of interest. Unsupervised learning methods with 400 documents yields field segmentation models of a similar quality to those learned using supervised learning with 50 documents.

## 12.3 Part-of-Speech Tagging in Context[3]

### 12.3.1 Overview

The method proposed in this paper tags a word using an HMM tagger that looks at context on both sides of a word, and is evaluated using both supervised and unsupervised methods. This paper provides the first comprehensive comparison of unsupervised POS tagging methods.

| | **Ads** | **Citations** |
|---|---|---|
| Baseline | 46.4 | 27.9 |
| Segment and cluster | 62.4 | 46.5 |
| Supervised | 74.4 | 72.5 |
| Unsupervised (learned trans) | 48.8 | 49.7 |
| Unsupervised (diagonal trans) | 70.0 | 66.3 |
| + Hierarchical (learned) | 70.1 | 39.1 |
| + Hierarchical (given) | 70.9 | 62.1 |
| + Boundary (learned) | 70.4 | 64.3 |
| + Boundary (given) | 71.9 | 68.2 |
| + Boundary + Hierarchical (learned) | 71.0 | - |
| + Boundary + Hierarchical (given) | 72.7 | - |

**Table 12.1.** Summary of results

Common POS taggers are formulated using HMMs, where the states correspond to POS tags ($t_i$) and words ($w_i$) are emitted when a state is visited. The training of HMM-based taggers involves estimating lexical probabilities, $\mathrm{P}(w_i|t_i)$, and tag sequence probabilities, $\mathrm{P}(t_i|t_{i-1},...,t_{i-n})$. Merialdo's tagger accounts for the previous two tags, i.e. $\mathrm{P}(t_i|t_{i-1},t_{i-2})$. This paper uses unsupervised transformation-based learning (UTBL), which looks at both left and right word contexts, not just the left context as previous HMM taggers do.

A traditional HMM tagger looks at the previous two tags with the word probability conditioned on the current tag. This ignores dependencies that may exist between a word and the POS tags of the words which precede and follow it. For example, verbs which associate with a particular POS but can also be tagged as nouns or pronouns (e.g. 'thinking that') may benefit from modeling dependencies on future tags.

The state transition probabilities model was stabilized via:

- contextual probabilities were initialized using trigrams where all words contained unambiguous tag sequences (all three words of the trigram contained at most one POS tag)
- transition model probabilities were trained while lexical probabilities were kept constant and uniform

The corpus used, the Penn Treebank, was split into sections for training,

| | **Unfiltered Lexicon** | **Optimized Lexicon** |
|---|---|---|
| Merialdo HMM | 71.9 | 93.9 |
| Contextualized HMM | 76.9 | 94.0 |
| Kupiec HMM | 77.1 | 95.9 |
| UTBL | 77.2 | 95.9 |
| Contextualized HMM with Classes | 77.2 | 95.9 |

**Table 12.2.** Unsupervised POS Taggers

| | **Test Accuracy** |
|---|---|
| Baseline | 92.19 |
| Standard HMM | 95.87 |
| Contextualized HMM | 96.59 |

**Table 12.3.** Supervised POS Taggers

development, and testing (sections 00 - 18 for training, 19-21 for development, and 22-24 for testing). To avoid the problem of unknown words, a lexicon constructed from tagged versions of the full Treebank was provided. Estimates of the likelihoods of tags for words were not provided, but only the knowledge of what tags are possible for each word in the lexicon (i.e. something that could be obtained from a manually-constructed dictionary).

A certain number of frequently occurring words (e.g. a, to, of) are sometimes labeled with infrequently occurring tags (e.g. SYM, RB). For the HMM taggers, which begin with uniform estimates of both the state transition probabilities and the lexical probabilities, the learner finds it difficult to distinguish between more and less probable tag assignments. Mistagging during Treebank construction also impacted tagging accuracy. The elimination of these noisy POS assignments raised accuracy back into the realm of previously published results.

### 12.3.2 Results

10,000 unique unambiguous tag sequences were extracted from the training data for initializing the state transition probabilities. One baseline tagger used for comparison chooses a word's most frequent tag. A standard HMM trigram tagger (which was trained) was implemented as another baseline for comparison.

## 12.4 Building Domain-Specific Taggers without Annotated Domain Data [4]

### 12.4.1 Overview

When a tagger is developed and trained in one domain (eg. Wall Street Journal articles) and then used in another domain (eg. medical domain) without being retrained, performance tends to degrade considerably. This degradation can be overcome by developing an annotated corpus for the new domain. But the problem with developing an annotated corpora is that it can be a very costly endeavor.

A primary cause of degraded performance when porting a tagger is that vocabulary is specific to the new domain, and hence unseen in the original domain. This paper focuses on handling domain-specific vocabulary by looking at suffix patterns.

Most taggers that use just suffix information require contextual information (tags of nearby words) in making their prediction. This requires a word dictionary for the domain being ported to. The paper's hypothesis is that words with similar suffix distribution will have similar POS tag assignments regardless of the domain. Looking at suffix patterns for predicting POS tags has been done before, but this paper is the first to apply it to the problem of domain adaptation. The kNN method [5] to associate words with possible POS tags in the new domain was used. Prefix information (apart from hyphenated words) were not incorporated in the study.

The tag assigned depends on context of use, which is computed using a first-order HMM. Transitional probabilities are computed by using smoothed WSJ-based probabilities which are adjusted to the new domain by using a domain-specific text in conjunction with Expectation Maximization (EM) training. Existence of annotated data is not assumed for the new domain.

The HMM is based on bigram modeling. The transitional probabilities correspond to $P(t_i|t_{i-1})$ where $t_i$ and $t_{i-1}$ are POS tags. Smoothed initial bigram probabilities are computed as

$$P(t_i|t_{i-1}) = \lambda P_{WSJ}(t_i|t_{i-1}) + (1-\lambda)P_{WSJ}(t_i) \tag{12.9}$$

where $\lambda = 0.9$.

The forward-backward EM algorithm was modified by dampening the change in transitional and emit probabilities for each iteration. The transitional probability is computed by

$$P(t_i|t_{i-1}) = \lambda P_{NEW}(t_i|t_{i-1}) + (1-\lambda)P_{OLD}(t_i|t_{i-1}) \tag{12.10}$$

where $P_{OLD}$ is the transitional probability of previous iteration and $P_{NEW}$ is the transitional probability of the forward-backward algorithm.

Previous methods achieved better accuracy by restricting the possible tags associated with words. Eliminating possibilities that rarely appear with a word reduces the chance that unsupervised training would spiral along an unlikely path. This paper's approach considerably reduces the number of tags to what is appropriate for each word. Further, any tag associated with low probability is removed by the kNN method (these tags are usually just noise introduced by some inappropriate exemplar).

The kNN method [5] was applied for handling words unseen in the training data. The estimated probabilities were used during the tagging process. Instead of just applying the method for unknown words (words not present in the training data) the approach of this paper is to create the entire lexicon in the new domain. But instead of using a supervised tagging model as in [5], this paper employs unsupervised learning with EM training. Suffix information was used to provide initial values, with the EM algorithm adjusting these initial probabilities to the new domain.

### 12.4.2 Developing The Lexicon And Assigning Initial Probabilities

A domain text, Text-Lex, was used to develop the initial lexical probabilities for the HMM. Let a word $w$ appear a sufficient number of times in Text-Lex (at least 5 times). Text-Lex was searched for related words in order to assign a feature vector to this word. The features are written as $-x+y$, where $x$ and $y$ represent either a suffix or the empty string (empty string represented as _).

The feature $-x+y$ represents the word formed by replacing some suffix $x$ in word $w$ by another suffix $y$ (i.e. $x$ is 'subtracted', and $y$ is 'added' to the word $w$'s stem). In the word 'creation', '-ion+' corresponds to the stem word 'create' with suffix 'ion' removed, and '-ion+ion' corresponds to the word 'creation' itself ('ion' is removed and then added back). The feature '-ion+ed' captures information about the word 'created', whereas the feature '-+s' corresponds to the word 'creations'. With a word like 'history', it might have non-zero values for '-y+ic' (historic) or '-+s' (histories), but it is likely to set a zero value for '-ory+' (unless 'hist' or 'histe' is found in Text-Lex). This zero value represents the fact that although 'history' has 'ory' as a suffix, it has no stem. Whether or not there is a stem bears much information for suffixes like 'ate' and 'ory'.

Suffix classes rather than actual suffixes are used since this provides a more appropriate level of abstraction. Given a word $w$ with suffix $x$ it is observed whether removing $x$ from $w$ leads to another word by using a few basic variations of English morphology. For a word with no suffix from our list of suffixes, $x$ is taken to be '_' i.e., empty string. For the suffix 'ed', replacing 'ied' with 'y' relates 'purified' with 'purify' and recognizes the spelling alternation of i/y. Thus for the word 'purify' the feature '-+ed' represents the presence of 'purified' since '+ed' represents the suffix class rather than the actual suffix. Similarly, removing a suffix is considered and, if necessary, adding an 'e' to see if such a word exists. This allows 'creation' to be related with 'create', or 'activate' with 'active'. Doubling of a few consonants is attempted to relate 'occurrence' and 'occur'. When a word could have two suffixes, the word is considered to always have the longer functional suffix. Hence, 'government' has the 'ment' suffix rather than the 'ent' suffix.

Two different types of vectors are used for any word: Bin (binary count) and RFreq (relative frequency). In the Bin vector associated with 'creation', all the four features from the above paragraph will get a value of one (assuming that the four corresponding words are found in Text-Lex). On the other hand, assuming 'creatory' is not found in Text-Lex, '-ion+ory' would get a zero value. For RFreq vector, instead of ones and zeros, the frequency of occurrences of each word is normalized so that the sum of all feature values is one. Thus a word with 4 features having non-zero frequencies of 10, 20, 30 and 40 will have the respective values set to 0.1, 0.2, 0.3 and 0.4. A word with four features having non-zero frequency, which are 1, 2, 3 and 4, will also have the same 4 relative frequency values. The intuition is that the Bin vector helps in determining the set of tags that associate with a word, and that the RFreq vector supplements this information with the likelihood of these tags. For example, a value of 1 for the '-ing+' feature in a Bin vector (thus disqualifying a word like 'during') may be sufficient to predict VBG, JJ and NN tags. However, this may not suffice to provide the ordering of likelihood among these tags for this word. On the other hand, it seems to be the case that when the 'ing' form appears far more often than the 'ed' form, then the NN tag is most likely. But if the 'ed' form is more frequent, then VBG is most likely. Examples in the WSJ corpus include 'smoking', 'marketing', 'indexing', and 'restructuring' for the first kind, and 'calling', 'counting', 'advising', and 'noting' for the second kind.

The Bin and RFreq vectors are created for the word $w$ based on the distribution of words in Text-Lex. Following the same method, the Bin and RFreq vectors for a word $v$ in the WSJ corpus are created by using the distributions in the WSJ corpus. BinDist($w$,$v$) can then be computed as the number of features in which the two Bin vectors differ. RFDist is similarly defined as a weighted sum of two distances: the first distance is the L1-norm distance based on feature values for which both words have non-zero values, and the second distance is based on values of features for which one word has a zero value and the other does not.

For example, if the two words' RFreq vectors are $< w_1, ..., w_n >$ and $< v_1, ..., v_n >$ then:

$$\text{RFsame}(w, v) = \sum_{w_i \neq 0 \cap v_i \neq 0} |w_i - v_i| \tag{12.11}$$

$$\text{RFdiff}(w, v) = \sum_{w_i = 0 \cap v_i \neq 0} |w_i - v_i| + \sum_{w_i \neq 0 \cap v_i = 0} |w_i - v_i| \tag{12.12}$$

$$\text{RFDist}(w, v) = \text{RFsame}(w, v) + \delta \cdot \text{RFdiff}(w, v) \tag{12.13}$$

For RFDist(), $\delta = 2$ was used. Given a word $w$, the 5 nearest neighbors from the WSJ corpus were found and their average lexical probabilities were used to obtain the lexical probabilities for $w$.

## Square Root Smoothing

Let $w$ be a word that has suggested tags $t_1, ..., t_n$ in order of likelihood ($t_1$ is most probable). Let Sqrt-score($t_i$) $= \sqrt{n + 1 - i}$. Now assign probabilities based on this score (after normalizing) so that the probabilities for the n tags sum to 1. For example, if a word $w$ has three possible tags, no matter what the original lexical probabilities were determined to be, if $t_1$ is determined to be most probable, then $P(t_1|w)$ will be 0.418 ($\frac{\sqrt{n+1-1}}{\sum_{i=1}^{3} \sqrt{n+1-i}}$, where $n = 3$), the second most probable tag will be assigned 0.341, etc. The intuition behind this square root smoothing method is that it may be appropriate for low frequency words, where empirical probabilities based purely on a kNN basis may not be entirely appropriate if the new domain is very different.

The drawback is that when there is sufficient information, it is lost by such flattening. Also lost is when a tag is significantly more probable for a word. For example, the word 'high' is mostly annotated as JJ in WSJ corpus, but RB and NN are also possible. Square root smoothing will flatten this distribution considerably.

### 12.4.3 Examples

Consider the word 'broaden'. If it was known that the following words were also in our lexicon, 'broadened', 'broadening', 'broadens' and 'broad', this would provide support in treating 'broaden' as a verb. The presence of a suffix morpheme suggests a POS tag (or a small set of POS tags) for the word. Most taggers use this info to predict tags for unknown words during the tagging process. The presence of the suffix can also indicate possible tags for the words it attaches to. For example, the morpheme 'ment' indicates 'government' is likely to be an NN and also 'govern' is likely to be a verb.

This information is specific to a language rather than a domain, so an annotated corpus in another domain is used to provide exemplars. For example, 'phosphorylate' (in the Biology domain) and 'create' (in the WSJ corpus) are similar in the sense both take on 'tion', 'ed', and 'ing' suffixes but not 'ly'. Since the WSJ corpus would provide POS tag information for 'create', it is used to support the treatment of 'phosphorylate'.

### 12.4.4 Results

#### Evaluating Construction Of The Lexicon

The paper's kNN method identifies 96.3% of the word/type pairs from the GENIA corpus. The paper's word/token lexicon construction was correct in 99.0% of the tags.

#### Evaluation Of The HMM Tagger

The more labelled data one has for training the better. It is a much better option than having to worry about unsupervised learning.

A 5-fold cross-validation is used on 2000 Medline abstracts (i.e. 5 partitions are formed and experiments conducted 5 times and results averaged). For each test partition, the remainder partitions are used for 'training' (this is unsupervised since EM is used and POS tag information associated with

| **POS Tagger** | % **Accuracy** |
|---|---|
| Paper's HMM (5-fold) | 95.77 |
| MedPost | 94.1 |
| PennBioIE | 95.1 |
| GENIA supervised | 98.26 |

**Table 12.4.** POS Taggers

the words is disregarded). The accuracy of the HMM (based on 5-fold cross-validation experiments) was 95.77%, obtained for the case where the lexical probabilities were taken directly from kNN using only RFDist and all tags assigned with probability less than 0.02 were discarded.

# Bibliography

[1] An Algorithm that Learns Whats in a Name. Bikel, Schwartz, Weischedel. Machine Learning Journal Special Issue on Natural Language Learning. 1999.

[2] Unsupervised Learning of Field Segmentation Models for Information Extraction. Grenager, Klein, Manning. Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05). 2005.

[3] Part-of-Speech Tagging in Context. Banko, Moore. 20th International Conference on Computational Linguistics (COLING), August 23-27, 2004.

[4] Building Domain-Specific Taggers without Annotated (Domain) Data. Miller, Torii, Vijay-Shanker. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). 2007.

[5] Language Independent Minimally Supervised Induction Of Lexical Probabilities. Cucerzan, Yarowsky. Proceedings of ACL-2000, Hong Kong. 2000.