A Handbook For Language Engineers

Ali Farghaly, Editor

February 12, 2003

CENTER FOR THE STUDY
OF LANGUAGE
AND INFORMATION

1

Statistical Natural Language Processing

Chris Callison-Burch and Miles Osborne

1.1 Introduction

Statistical natural language processing (SNLP)[1] is a field lying in the intersection of natural language processing and machine learning. SNLP differs from traditional natural language processing in that instead of having a linguist manually construct some model of a given linguistic phenomenon, that model is instead (semi-) automatically constructed from linguistically annotated resources. Methods for assigning part-of-speech tags to words, categories to texts, parse trees to sentences, and so on, are (semi-) automatically acquired using machine learning techniques.

The recent trend of applying statistical techniques to natural language processing came largely from industrial speech recognition research groups at AT&T's Bell Laboratories and IBM's T.J. Watson Research Center. Statistical techniques in speech recognition have so vastly outstripped the performance of their non-statistical counterparts that rule-based speech recognition systems are essentially no longer an area of research. The success of machine learning techniques in speech processing led to an interest in applying them to a broader range of NLP applications. In addition to being useful from the perspective of producing high-quality results, as in speech recognition, SNLP systems are useful for a number of practical reasons. They are cheap and fast

[1]The authors would like to thank Jason Baldridge, Stephen Clark, Beata Kouchnir, Jochen Leidner, and Sarah Luger for their thorough comments on this chapter.

A Handbook For Language Engineers.
Ali Farghaly, Editor.

to produce, and they handle the wide variety of input required by a real-world application. SNLP is therefore especially useful in industry. In particular:

- SNLP affords rapid prototyping. Whereas fully hand-crafted systems are extremely time consuming to build, statistical systems that are automatically trained using corpora can be produced more quickly. This allows many different approaches to be tried and evaluated in a short time-frame. As an example, Cucerzan and Yarowsky described how one might create a new part-of-speech tagger in a single day (Cucerzan and Yarowsky, 2002). An even more ambitious example is Al-Onaizan et al.'s "machine translation in a day" experiment wherein they used statistical techniques to develop a complete Chinese-English machine translation system in a 24-hour period (Al-Onaizan et al., 1999).
- Statistical systems are "robust" (Junqua and van Noord, 2001). Although this has a wide variety of meanings, in SNLP it generally means that a system will always produce some output no matter how badly formed the input is, and no matter how novel it is. For example, a text classification system may be able to classify a text even if all of the words in that text are previously unseen. Handling all kinds of input is necessary in real-world applications; a system which fails to produce output when it is unable to analyze a sentence will not be useful.
- Statistical systems are often cheaper to produce than hand-crafted rule-based systems. Because the process of creating a statistical system is more automated than the process of creating a rule-based system, the actual number of participants needed to create a system will often be less. Furthermore, because they are learned from data, statistical systems require less knowledge of the particular language being analyzed. This becomes a budgetary issue on a multi-language project because of the expense of hiring language consultants or staff with specialized skills.

A common theme with many early SNLP systems was a pride in minimizing the amount of linguistic knowledge used in the system. For example, Fred Jelinek, the then leader of IBM's speech recognition research group, purportedly said, "Every time I fire a linguist, my performance goes up." The sentiment is rather shocking. Should Jelinek's statement strike fear into the hearts of all linguists reading this chapter? Is there a strong opposition between theoretical linguistics and SNLP? Will SNLP put linguists out of work?

We put forth a positive answer in this chapter: there *is* a useful role

for linguistic expertise in statistical systems. Jelinek's infamous quote represents biases of the early days of SNLP. While a decade's worth of research has shown that SNLP can be an extremely powerful tool and is able to produce impressive results, recent trends indicate that using naive approaches that are divorced from linguistics can only go so far. There is therefore a revival of interest in integrating more sophisticated linguistic information into statistical models. For example, language models for speech recognition are moving from being word-based "n-gram" models towards incorporating statistical grammars (Chelba and Jelinek, 1998, Charniak, 2001). So there is indeed a role for the linguist. This chapter will provide an entry point for linguists entering into the field of SNLP so that they may apply their expertise to enhance an already powerful approach to natural language processing.

Lest we represent SNLP as a completely engineering-oriented discipline, we point the interested reader to Abney (1996) which describes a number of ways in which SNLP might inform academic topics in linguistics. For example, SNLP can be useful for psycholinguistic research since systems typically encode graduated notions of well-formedness. This offers a more psychologically plausible alternative to the traditional binary grammatical/ungrammatical distinction. In a similarly academic vein, Johnson (1998) shows how Optimality Theory can be interpreted in terms of statistical models. This in turn suggests a number of interesting directions that OT might take.

The rest of this chapter is as follows: We begin by presenting a simple worked example designed to illustrate some of the aspects of SNLP in Section 1.2. After motivating the usefulness of SNLP, we then move onto the core methods used in SNLP: modeling, learning, data and evaluation (Sections 1.3, 1.4, 1.5, and 1.6 respectively). These core methods are followed by a brief review of some of the many applications of SNLP (Section 1.7). We conclude with a discussion (Section 1.8) where we make some comments about the current state of SNLP and possible future directions it might take.

1.2 A Simple Worked Example

Imagine we have just founded a start-up company, and want to automate the task of replying to customer email queries. Further imagine that we decide to do this by parsing each sentence in the messages, carrying out some sort of semantic interpretation step (for example, by querying a database of customer orders) and then on the basis of this interpretation, automatically generate an appropriate reply.

Assuming that we have a grammar in place, one problem that would

need to be addressed by our start-up company is ambiguity. A question as simple as "Do you sell Apple laptops?" has multiple parses (two are shown in Figure 1) corresponding to multiple interpretations. The question could either be asking:

(a)whether the company stocks Apple-branded laptops, or

(b)whether the company sells laptop computers to Apple.

For sentences which have multiple parses, how do we know which is the most likely interpretation? Given that all of these parses are possible in some context or other, we are faced with the problem of having to select the most likely parse. (Sounds like the perfect application of probability, no?)

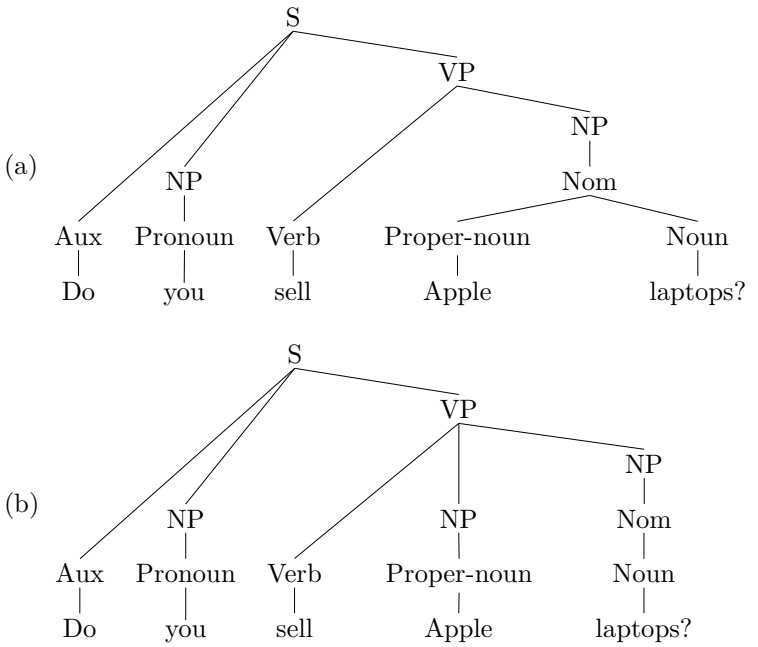

FIGURE 1 Two possible parses (T_1 and T_2) for "Do you sell Apple laptops?".

Let's assume for the moment that our start-up company is using context-free grammars (CFGs) to do the parsing. Also, let's assume that the sentences within a message are independent of each other.[2]

[2]This assumption is clearly wrong – the meaning of one sentence in a message

CFGs are defined by a set of rules where each rule is of the form $A \rightarrow \beta$, where A is a non-terminal symbol and β is a sequence of terminal and/or non-terminal symbols. Probabilistic context-free grammars (PCFGs) extend CFGs by associating each rule with a probability:

$$A \rightarrow \beta \ [p]$$

A simple grammar is shown in Figure 2 (adapted from Jurafsky and Martin (2000)).

S → NP VP	[.80]	Det → that	[.05]
S → Aux NP VP	[.15]	Det → the	[.80]
S → VP	[.05]	Det → a	[.15]
NP → Det Noun	[.20]	Noun → laptops	[.50]
NP → Proper-Noun	[.35]	Noun → desktops	[.50]
NP → Nom	[.05]	Verb → sell	[.30]
NP → Pronoun	[.40]	Verb → ship	[.55]
Nom → Noun	[.75]	Verb → repair	[.15]
Nom → Noun Noun	[.20]	Aux → can	[.60]
Nom → Proper-Noun Noun	[.05]	Aux → do	[.40]
VP → Verb	[.55]	Proper-Noun → Apple	[.50]
VP → Verb NP	[.40]	Proper-Noun → Sony	[.50]
VP → Verb NP NP	[.05]	Pronoun → you	[.40]
		Pronoun → I	[.60]

FIGURE 2 A very simple probabilistic grammar for English

The probability of a rule is the likelihood that a sequence of terminals or non-terminals β will occur given that they are immediately dominated by a non-terminal symbol A. This is written as the conditional probability $P(A \rightarrow \beta|A)$. Having probabilities associated with the application of each rule is useful for disambiguation, because they can be combined to calculate the probability of a parse tree as a whole. Competing parses for a single sentence can then be compared against each other using this probability to see which parse is most likely (and thus preferred). The probability of a particular parse T can be defined as the product of the probabilities of every rule $A \rightarrow \beta$ used to expand each node A in the parse tree:

$$P(T) = \prod P(A \rightarrow \beta|A)$$

is not unrelated to the sentences that precede it – yet is indicative of the kinds of simplifying assumptions made in practical SNLP systems.

For example, the probability of the tree corresponding to the question about whether the e-commerce company stocks Apple laptops would be:

$$\begin{aligned}
P(T_1) =& P(\text{S} \rightarrow \text{Aux NP VP} \mid \text{S}) * P(\text{Aux} \rightarrow \text{Do} \mid \text{Aux}) * \\
& P(\text{NP} \rightarrow \text{Pronoun} \mid \text{NP}) * P(\text{Pronoun} \rightarrow \text{you} \mid \text{Pronoun}) * \\
& P(\text{VP} \rightarrow \text{Verb NP} \mid \text{VP}) * P(\text{Verb} \rightarrow \text{sell} \mid \text{Verb}) * \\
& P(\text{NP} \rightarrow \text{Nom} \mid \text{NP}) * P(\text{Nom} \rightarrow \text{Proper-Noun Noun} \mid \text{Nom}) * \\
& P(\text{Proper-Noun} \rightarrow \text{Apple} \mid \text{Proper-Noun}) * \\
& P(\text{Noun} \rightarrow \text{laptops} \mid \text{Noun}) \\
=& .15 * .40 * .40 * .40 * .40 * .30 * .05 * .05 * .50 * .50 \\
=& 7.2 * 10^{-7}
\end{aligned}$$

In comparison, the probability of the tree corresponding to the question as to whether the e-commerce company sells computers to Apple would be:

$$\begin{aligned}
P(T_2) =& P(\text{S} \rightarrow \text{Aux NP VP} \mid \text{S}) * P(\text{Aux} \rightarrow \text{Do} \mid \text{Aux}) * \\
& P(\text{NP} \rightarrow \text{Pronoun} \mid \text{NP}) * P(\text{Pronoun} \rightarrow \text{you} \mid \text{Pronoun}) * \\
& P(\text{VP} \rightarrow \text{Verb NP NP} \mid \text{VP}) * P(\text{Verb} \rightarrow \text{sell} \mid \text{Verb}) * \\
& P(\text{NP} \rightarrow \text{Proper-Noun} \mid \text{NP}) * \\
& P(\text{Proper-Noun} \rightarrow \text{Apple} \mid \text{Proper-Noun}) * \\
& P(\text{NP} \rightarrow \text{Nom} \mid \text{NP}) * P(\text{Nom} \rightarrow \text{Noun} \mid \text{Nom}) * \\
& P(\text{Noun} \rightarrow \text{laptops} \mid \text{Noun}) \\
=& .15 * .40 * .40 * .40 * .05 * .30 * .35 * .50 * .05 * .75 * .50 \\
=& 4.725 * 10^{-7}
\end{aligned}$$

According to the probabilistic grammar in Figure 2, $P(T_1) > P(T_2)$. We might conclude that the first tree is preferred over the second tree, and would therefore come to the conclusion that the customer wanted to know whether we sold laptops made by Apple.

A natural question to ask is where did the probabilities that are assigned to each grammar rule come from? While we could have manually assigned them (perhaps by inspecting parse trees and looking for likely combinations of rule applications), this approach would be time-consuming and error-prone. Instead, the probabilities associated with

CFG rules are usually automatically set through a process called *estimation*. Section 1.4 reviews some of the popular estimation techniques.

Finally, we did not really justify why we tied probabilities to context-free rules rather than tying them to other aspects of the tree. Other *models* of parse selection are also possible. For example, better performance can be achieved if we also include greater syntactic context into rules; if we add information about the lexical head of a given phrase then performance improves again. The process of specifying how some task is going to be tackled is known as *modeling*.

1.3 Modeling

SNLP, like linguistics and science in general, can be thought of as building theories and testing them. For example, if we were interested in seeing whether our latest theory of long distance dependencies helped in question answering, we would create a *model* of how questions are answered and incorporate information about long-distance dependencies into that model. We would then test the model by evaluating the performance of the question answering system. Modeling is the general task of constructing machinery which mimics some task. The process of building a statistical model takes place in a number of stages:

1. First, the structural components of a model are constructed. This usually takes place manually, but can take place automatically for some kinds of tasks such as text classification through clustering documents by similarity. If we were building a model of syntax, we would first enumerate a set of possible grammar rules. In non-statistical linguistics, this step would be the entirety of the modeling process. It would be up to the linguist to write appropriately constrained rules so that no ill-formed sentences would be admitted. In statistical approaches, however, models also include a probabilistic part in addition to structural components.
2. Next, the model is parameterized. Parameters can be thought of as variables that take probabilities as values. The parameters of a model of grammar would correspond to the rule probabilities. The probabilities would specify how the rules combine together. This would free the linguist from having to exactly constrain a grammar in order to eliminate all spurious parses, by letting the probabilities automatically select preferred parses.
3. Finally, the model is instantiated by assigning values to the parameters. Instantiating a model of grammar would amount to specifying probabilities for the application of each of the rules. Probabilities are usually estimated from linguistically annotated

data, such as a treebank (sentences annotated with their parse trees).

The interaction of how a model is parameterized and how its values are estimated is a significant concern. If we specify a model that is too complex (has too many parameters to be estimated), then we will model the *training set* (samples of the phenomena we are interested in modeling) too closely, and so fail to adequately predict future examples of that phenomena. In the extreme, we would consider that the training set *is* the world and so completely fail to generalize to new situations. This failure to generalize beyond the training data is called *overfitting*. An extreme example grammar that overfitted would be one that had a single rule per sentence. A more natural overfitted example would be a model that only contained syntactic constructs found in the treebank. If we constructed a grammar by extracting only those rules found in our treebank, and if some examples of some class of ambiguities (such as PP attachment) were not contained in it, then an overfitted model would not consider them at all. Overfitting is a central concern to us as we never have enough (annotated) training material and so always run the risk of basing our decisions upon insufficient evidence.

In contrast, if our model overgeneralizes, then again performance at some task will suffer. We will incorrectly model malformed examples. An example grammar that would overgeneralize would contain rules which generated any string possible. This is called *underfitting*. Underfitted models are too simple. When modeling, we therefore need to strike the correct balance between simplicity and complexity. Generally, this trade-off is carried out manually, by a process of trial-and-error. We will return to this problem later when we talk about *smoothing* (Section 1.4.1).

In addition to overfitting and underfitting, another fundamental aspect of modeling is whether a model should be *generative* or *discriminative*. This depends on whether we are concerned with building up machinery which could create language or whether part of the language is already given (observed) in which case the task would be to discriminate between alternative structures. *Generative* models, as the name implies, are concerned with actually creating structures. Generative models are natural ways of thinking about many tasks. Grammars can be thought of as generative models in that they can be used to generate sentences. *Discriminative* models instead try to distinguish between a set of possibilities when something is already given. In particular, they do not fully model the task. Machine translation can be thought of as a discriminative task. Given a sentence in the source language we want

to choose the best possible translation from a set of sentences in the target language. Within a discriminative approach the source sentence is not modeled.

Generative and discriminative models also have a fundamental difference in how they relate to probability theory. If we consider a generative model of part-of-speech (POS) tagging, then we are interested in the joint probability $P(\text{words}, \text{tags})$. This generative model relates sentences to their POS tag sequences (and vice-versa). It also models the (marginal) distributions of sentences and POS tag sequences in isolation of each other. One problem with generative (joint) approaches is that we need to deal with the distributions of all points in the model. A discriminative model $P(\text{tags} \mid \text{words})$ does not model the (marginal) distribution of words – $P(\text{words})$ – and is therefore simpler than the joint probability $P(\text{words}, \text{tags})$. Discriminative approaches model the choices we can make given some context, and do not waste effort on the context itself. Joint and discriminative models are currently under intense investigation (Johnson, 2001, Lafferty et al., 2001).

1.4 Estimation / Learning

Once we have worked out how to model some task and which parameters to use, the next step is to assign values to the parameters. This step is called *estimation* (if you are statistically-minded) or *learning*. Here, we will use the term *estimation*. Once we have written down a set of grammar rules, we then want to estimate the probabilities of each of the rules using some training data. There are a number of techniques that can be used to estimate probabilities from data. Techniques can be distinguished by properties such as bias and scalability:

- *Bias* has a variety of meanings, but typically it can be thought of as some systematic preference for some type of model instantiations. Bias in estimation is usually present either by design, or else as some property of the process itself. Occam's razor is one example of bias by design (towards simple models); Epicurus's principle of multiple explanations ('if more than one model is consistent with the data, keep all models') is another. Examples of bias in design are maximum likelihood estimators (Section 1.4.1) which are biased towards models that give maximal consideration to the training set, and maximum entropy models (Section 1.4.2) which are biased towards models that are probabilistically simple.
- *Scalability* is a practical concern about whether an estimation method can be used with a large number of parameters, or with limited computational resources. If we have to associate one parameter per word

then we need an estimation technique which will scale appropriately. Also, we want the estimation method to be fast enough to train so that we can run it again and again when testing different possible models. Clearly, this enterprise will only work if we can deal with hundreds of thousands of parameters, and in reasonable time.

There are a large number of estimation methods that can be used to assign values to the parameters of a model. Here, we briefly look at some of the more popular approaches. We concentrate on supervised methods, which assume the training material is annotated in some manner, but also turn to weakly supervised methods (Section 1.5.1), which use both annotated and unannotated training data.

1.4.1 Maximum likelihood and smoothing

By far the most popular estimation method in SNLP is *maximum likelihood estimation* (MLE). A maximum likelihood estimator finds a set of parameters such that the probability of the training set, given a model, as high as possible. As such, it has a firm statistical grounding. However, the real reason for its popularity is the fact that the parameters can be estimated by simple counting. For example, if some word takes a given part-of-speech tag n times, and the sum of all possible part-of-speech tags for that word is N, then if we make the (incorrect) assumption that words are independent of each other, the maximum likelihood estimate of the probability that the word takes a given part-of-speech tag would be the ratio $\frac{n}{N}$. The probability of a grammar rule applying to a particular word could be estimated by counting the number of times it applies to the word divided by the total number of rules applied to that word in a treebank.

MLE is biased towards the training set. When the training set contains errors (which is often referred to as "noisy" data) or when the training set is not representative of examples we want to classify in the future, the models produced using MLE need to be modified: for typical language training sets, models estimated using MLE tend to overfit. Usually, these models are *smoothed* in an attempt to improve performance. Good smoothing is central to successful MLE. For example, Collins and Brooks showed how a simple model of PP-attachment, estimated using MLE, could be greatly improved by smoothing (Collins and Brooks, 1995). Popular smoothing methods include backing-off, linear interpolation and Good-Turing smoothing and combinations thereof (Chen and Goodman, 1996). Successful smoothing is a 'black art' and generally involves a large degree of engineering. Note that performance could also be increased by selecting some other, possibly simpler model,

and then applying MLE without smoothing.

1.4.2 Maximum entropy

Counting-based MLE can be acceptable when the model explicitly captures dependencies within some task. However, sometimes dependencies are not obvious and therefore cannot easily be encoded into a model. Instead of enumerating the points of dependencies explicitly, one could use an estimation technique which automatically accounts for dependencies when they exist. *Maximum entropy* (maxent) models – also called log-linear models – are capable of correctly estimating the parameters for modeling data which may contain dependencies (Abney, 1997). Maxent is a form of MLE, but the parameters are calculated using iterative numerical optimization methods. Maxent has been applied to almost all SNLP tasks, usually producing competitive results (Ratnaparkhi, 1998).

Maxent, like counting-based MLE, is also prone to vagaries of the training set. While it does have a preference for simplicity, it can produce suboptimal results, especially when dealing with very sparse data.[3]

Maxent models are estimated using numerical optimization methods. Previously we mentioned that scalability is an important consideration when building large models. Malouf showed that for a range of language tasks, general-purpose constrained optimization techniques such as limited memory variable metric approaches converged much faster than special-purpose maxent estimation methods such as Generalized or Improved Iterative Scaling (Malouf, 2002).

1.4.3 Kernel methods

The previous two methods assumed some model abstracted from the training set, with associated parameters that need to be estimated. The idea then is that the instantiated model is a summary of the training set. *Kernel methods* (such as Support Vector Machines) take an alternative approach. The 'model' is determined in terms of a number of key training examples which, when combined together in a functional form, are able to act as a classifier. These training items are usually selected such that the resulting classifier maximally separates the training examples. Frequently, this separation is generally not possible in the original feature space, and so the training examples are transformed, via a kernel function, into another space, whereby the training examples can be linearly separated. Kernel methods are attractive since they can capture non-linear interactions between features (maxent models

[3]Recent work has addressed this problem using a Gaussian prior (Chen and Rosenfeld, 1999).

assume features are related in a log-linear manner). Kernel methods have been applied to an increasing range of language tasks, usually with (near) state-of-the-art performance (Joachims, 1998).

As with maxent models, kernel methods require numerical optimization. Current estimation methods for SVMs do not scale well and therefore properly training them is infeasible for some applications. For example, when classifying phones using the TIMIT corpus, Salomon et al. (2002) estimated that it would take six years of CPU time to estimate the SVM model. There are various extensions and approximations which help reduce this burden.

1.4.4 Ensemble approaches

One of the findings over the last decade has been the fact that performance can often be increased by combining different models into an ensemble. Multiple models can be combined simply by having each model vote on the classification being made, say a part-of-speech tag. The ensemble would then output the tag which received the most votes. Other options include weighting each constituent model's vote by how well it does on some testing set, and so forth.

Ensemble techniques (such as *boosting*, *bagging* and *stacking*) are currently popular (Dietterich, 2000). The basic idea here is that all models will be biased in some manner, and so this unwanted bias can be eliminated by averaging over a number of alternative models. In the language domain, examples of ensemble approaches have been applied to POS tagging (Brill and Wu, 1998), parsing (Henderson, 1998) and text classification (Ghani, 2000). In virtually all cases, ensemble techniques produce the best results. The price to pay here is that ensemble methods greatly increase the computational burden associated with estimation. For example, instead of estimating a single model, we now need to estimate possibly hundreds of models. Furthermore, ensemble models push complexity to the extreme and are very hard to interpret.

1.5 Data

SNLP is a data-intensive field. Because it uses machine learning to automatically estimate a model's parameters from data, the success or failure of an undertaking can depend on the quality and availability of appropriate data. The data used in computational linguistics tasks generally takes the form of corpora. Corpora can be divided into two categories: annotated corpora and unannotated corpora. Unannotated corpora are simply large collections of raw text. They are useful for tasks such as n-gram language modeling which rely on estimating the frequency of words, bigrams, and trigrams, or for context-sensitive

spelling correction. Annotated corpora add additional information to the text, such as phonetic transcription, part-of-speech tags, parse trees, rhetorical relations, etc.

In the early days of computational linguistics, statistical approaches were limited by the availability of machine-readable text. Now, the heavy increase in use of the internet over the past decade has greatly expanded the amount of easily accessible machine-readable text. There have been a number of interesting proposals on how to exploit the web as a data source. For example, Resnik (1999) describes a method for mining the web for bilingual texts, which are vital for statistical machine translation. The method automatically gathers web pages that are potential translations of each other by looking for documents in one language which have links whose text contains the name of another language. For example if an English web page had a link with the text "Español" or "en Español" then the page linked to is treated as a candidate translation of the English page into Spanish.

While the web offers a promising source of data for a few tasks, most natural language processing tasks require data of a very specialized sort. Tasks as simple as part-of-speech tagging or as complicated as parsing require a corpus of linguistically annotated examples. Figure 3 describes a number of annotated corpora that are commonly used in computational linguistics research. Most are available through the Linguistics Data Consortium[4]. The availability of an appropriate annotated corpus greatly facilitates statistical NLP research into a particular task. For example, the Penn Treebank is an invaluable resource for statistical parsing. The syntactic structures associated with each sentence in the Penn Treebank could be used to estimate the probabilities of rules for the statistical context free grammar introduced earlier, which would then allow us to perform the disambiguation on e-mail response.

The creation of a resource as large as the Penn Treebank is costly, time-intensive and often tedious. Marcus et al. (1993) give a detailed description of the process. The high cost of creating an annotated corpus is largely to do with having to pay salaries of appropriately skilled laborers (such as linguistics graduate students). Depending upon the amount of data required, the availability of staff and the amount of error checking carried out, the process can take years. The Penn Treebank itself was developed over a three year period using a team of four annotators, all with graduate training in linguistics. Marcus et al. (1993) estimated that a team of five part-time annotators annotating three hours a day should be able to produce an output of about 2.5

[4] `http://www.ldc.upenn.edu/`

Corpus name	Words	Description
Brown Corpus	1M	The first modern, computer readable corpus. Consists of various texts in American English.
BNC	100M	A large corpus of spoken and written British English, completely annotated with part-of-speech tags.
Penn Treebank	1M	Wall Street Journal sentences annotated with parse trees.
CHILDES	various	A collection of child language data corpora.
Switchboard	3M	Transcribed telephone conversations and spoken texts. Includes recordings of the sentences used in the Penn Treebank.
HCRC Map Task	145K	Audio, video, and transcriptions of spoken dialogue between individuals participating in a cooperative task.
Canadian Hansard	20M	Bilingual sentence-aligned French and English proceedings of the Canadian parliament.

FIGURE 3 Various commonly used corpora

million words a year of parse-annotated sentences, based on an (optimistic) average rate of 750 words per hour. This estimate does not factor in any additional time for verifying the parses.

Research into SNLP is largely guided by the availability of appropriately annotated resources. Particular tasks, such as statistical parsing, tend to get more attention than other problems such as semantic interpretation simply because corpora annotated with parse trees exist. Corpora annotated with logical forms, on the other hand, tend not to exist in sufficient quantities. As such, research into statistical semantic interpretation is far less developed.

1.5.1 Weakly supervised methods

Given the central role of annotated material and the high costs associated with creating new annotated resources, recent research has focused on bootstrapping larger amounts of training data from existing resources. One approach is to combine relatively small quantities of manually annotated material with much larger volumes of unannotated material, using methods such as *co-training*. Co-training can be informally described in the following manner (Blum and Mitchell, 1998, Yarowsky, 1995):

- Pick two (or more) "views" of a classification problem.
- Build separate models for each of these "views" and train each model on a small set of labeled data.
- Sample from an unlabeled data set to find examples that each model independently labels with high confidence (Nigam and Ghani, 2000).
- Take these examples as being valuable training examples and iterate this procedure until the unlabeled data is exhausted.

Effectively, by picking confidently labeled data from each model to add to the training data, one model is labeling data for the other. Co-training differs from iterative re-estimation techniques in the Expectation Maximization (EM) family, since co-training is greedy in relying on the addition of confidently labeled material to re-estimate and gather new parameters, rather than re-estimating all parameters in each iteration. This greediness helps prevent co-training from falling into a suboptimal local minimum.

Co-training has been used for applications such as word-sense disambiguation (Yarowsky, 1995), web-page classification (Blum and Mitchell, 1998), named-entity identification (Collins and Singer, 1999), statistical machine translation (Callison-Burch, 2002) and parsing (Steedman et al., 2003). Figure 4 shows the performance of a statistical parser as it is co-trained with another statistical parser. Although

both parsers were initially trained with parse trees from the Brown corpus (non-newswire material), evaluation is in terms of performance at predicting parses for another domain, namely newswire material (the Penn Treebank). The unlabeled examples were raw newswire sentences. The higher curve shows what happens when a tiny amount of domain-specific annotated training material is added to the much larger volume of unannotated material from a different domain.

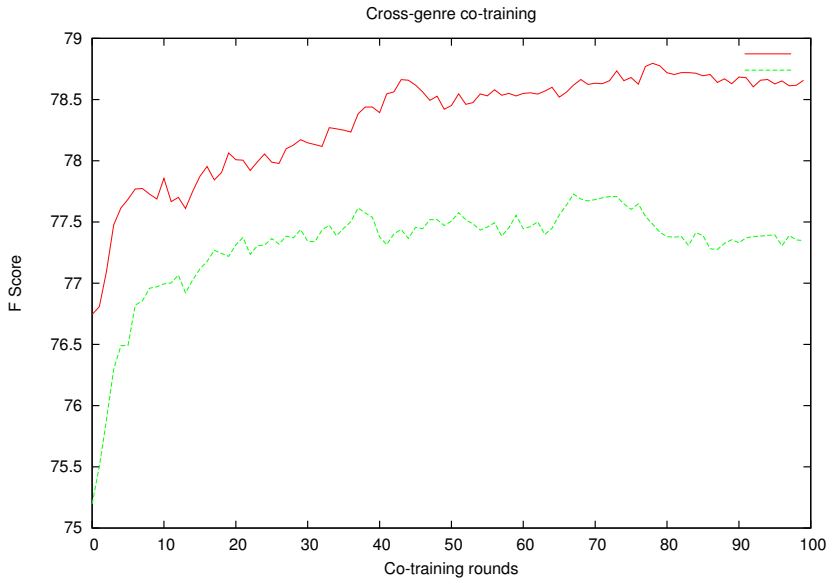

FIGURE 4 Cross-genre bootstrapping results from Steedman et al. (2003). The upper curve is for 1000 sentences labeled data from Brown plus 100 WSJ sentences, while the lower curve only uses 1000 sentences from Brown. The F-Score is a measure of the precision and recall of the parsers.

Another approach is to involve people in the annotation task (apart from the act of creating the initial seed set of annotated data, co-training is fully automatic). *Active learning* is similar to co-training, except that when two (or more views) all predict different annotations for some example, that example is labeled by a person. Otherwise, the system automatically annotates examples. Active learning (in the language community) has been applied to the task of building parsers and taggers.

1.6 Evaluation

One of the most critical parts of any NLP task (be it statistical or non-statistical) is having a clearly defined evaluation metric. This metric is used to judge how successful a particular method is by comparing it against the performance of other methods. Since it might not always be obvious which method is best for a given task, having a defined evaluation metric allows alternative approaches to be quantitatively compared with each other.

For example, one widely studied task is text classification. Most newly developed machine learning techniques will be applied to text classification as a standard benchmark task. The evaluation metrics used for text classification (as well as for many other tasks) are *precision* and *recall*. In the context of text classification, precision is the number of correctly classified documents divided by the number of classifications posited. Recall is the number of documents assigned a particular classification divided by the number of documents in the testing set which actually belong to that class. Precision and recall are complementary measures of accuracy, and can vary with each other. For example, if a text classifier assigned a classification to only one document, but got that document's classification correct, it would have a precision of 100%, but would have a very low recall if there were a lot of instances of documents of that class. Similarly, if a text classifier assigned the same classification to all documents it would have a recall of 100% for that label (because it correctly labeled every instance of class), but a very low precision (because it incorrectly labeled instances of other classes as that class). Determining the precision and recall for one machine learning technique allows it to be compared to others. Furthermore, in order for two machine learning techniques to be directly compared they need to be compared on identical tasks.

Yang and Liu (1999) compared the effectiveness of various machine learning techniques (including support vector machines, k-nearest neighbors, neural networks, linear least-squares fit mapping, and naive Bayes) on the same text classification task. They trained systems to perform "topic spotting" using identical newswire articles. Because each of the techniques was trained on an identical set of data, and further because they were tested on their ability to categorize the same set of articles their resulting scores were directly comparable. In cases where the difference in scores was very close Yang and Liu (1999) performed significance tests to determine whether the differences were due to better performance or to random factors. Though significance testing is not frequently done in natural language processing research

(often because one method obviously outperforms another, or else the assumptions behind hypothesis testing are violated), it can be useful in experimental methodology. Interested readers are referred to Dietterich (1998) for further discussion.

1.6.1 Training and testing sets

The notion of *training* and *testing* sets is important. Because of the "learned from data" nature of statistical natural language processing applications, training and testing data must be disjoint. A common pitfall is to train on data which is included in the evaluation set. This will result in falsely high performance estimates. The reason that performance estimates may be falsely high is because of the potential for overfitting the testing data. If the testing data is included in the training data then a machine learning technique may learn how to classify those examples exactly, but while doing so may not generalize well to unseen data.[5]

In situations where there is only a small amount of data available for training it may be difficult to divide the data into training and testing sets such that the testing set will fairly reflect the data as a whole. In such cases people often use cross validation. In cross validation the data is randomly divided into n sections. The learner is trained from $n-1$ of these sections and then evaluated against the remaining one. This is done n times, and the performance of the system is reported as the average of the ten evaluations. Note that reporting performance using cross-validation reduces the need for significance testing as chance variation in performance is accounted for through the averaging process.

1.6.2 System evaluation

A clearly defined evaluation metric does not only help in the comparison of different machine learning algorithms, but also in the comparison of full-blown NLP systems. In many situations it is useful to be able to evaluate the performance of various systems for a particular task. For instance, in industry you might be integrating other people's software into your own NLP applications. When deciding which commercial software package to purchase, or whether to purchase software rather than using an open source alternative, it is important to get an idea of the relative accuracy of the various systems. In such cases it will not be a matter of dividing a set of labeled data into a testing set, since the

[5] In a similar vein, it is common practice to further divide the training data by separating out a *held-out set* to help counteract overfitting. A learning algorithm will optimize its performance on the training data until the performance on the held-out begins to fall. When this happens one may assume that the algorithm is failing to generalize and is instead overfitting.

systems will likely be trained already, and the training data may not be distributed with the system.

When evaluating a commercial system it may prove useful to develop a *baseline* system to compare it against. A baseline system is usually the simplest implementation that one can think of. For instance, in part-of-speech tagging a baseline system would be one that always assigns the most frequent tag given a particular word without attempting to do any contextual disambiguation. Having a baseline system allows a reference point which can determine how difficult the task is. If a baseline system performs well, the task may be easy in which case you might not need to license a commercial system. If a system does not do much better than the baseline, it could either be because the task is very easy, the so-called baseline system is not really trivial, or else because the system is not very good (which should again give you pause about licensing it).

Finally, it is useful to gauge the *upper* and *lower* bounds upon performance. The upper bound is usually human performance at the task, whilst the lower bound might be chance. The baseline performance lies between these two extremes.

1.6.3 Component evaluation and error analysis

When evaluating a third-party system it is often only possible to do a "black box" evaluation wherein a system is just evaluated based on its performance as a whole. However, when developing your own system or when combining other systems it is possible to design evaluation metrics for each piece of the system. This will allow weak points to be identified, and expose parts that would be most effective to upgrade.

When performing component evaluation or when evaluating one's own system, it is often fruitful to see exactly which cases it is getting wrong, and to try to identify why. Performing such an *error analysis* is useful because it can often lead to indications about how the system may be improved. Performing an error analysis is one stage in the development of a SNLP system in which a linguistics background comes in especially handy. Being able to inspect the types of errors that are being made, and being able to generalize them into linguistic features is extremely useful for subsequent redesign of the statistical model used by the system.

1.7 Applications

SNLP is not just traditional natural language processing (or computational linguistics) augmented with probabilities. There are some applications which are more naturally handled with statistical techniques.

Relatively simple statistical methods can often be used in place of more involved rule based theories. Here, we quickly review a representative set of applications from a SNLP perspective.

1.7.1 Part-of-Speech Tagging

One of the early successes of SNLP was part-of-speech (POS) tagging. The basic task here is to assign a label (from a set of POS tags) to each token encountered. Figure 5 shows an example sentence with POS tags. The sentence is from the Parsed Wall Street Journal and the tags are from the Penn tagset.

The_DT stores_NNS they_PRP work_VBP for_IN may_MD be_VB sold_VBN

FIGURE 5 Example sentence with POS tags

POS tagging is useful for most NLP tasks especially information extraction, shallow parsing and full parsing.

There are many ways to tag sentences. The most popular method is to take a large corpus of sentences marked with tags (such as the Penn Treebank) and then train a model upon those tagged sentences. More formally, if S is a sentence and T is the associated tag sequence, we construct a model for the distribution $P(T, S)$. This is a joint probability. If all we care about is assigning tags to sentences, then we could instead construct a model for the conditional probability $P(T \mid S)$. Typically, *Hidden Markov Models* (HMMs) are used to construct tagging models (see the speech chapter for a discussion of HMMs). However it should be noted that a range of other methods are also possible. Current taggers operate at around 96% per-token accuracy.

Taggers can also be built by hand. For example, Samuelsson and Voutilainen (1997) argue that a manually built tagger can be competitive with its stochastic cousins. However, manually creating a tagger requires much more effort than creating it automatically, assuming that a corpus exists.

Why does POS tagging have such high accuracy? There are a number of reasons why this is so. Closed-class words – which are frequent – are usually unambiguous. Of the open-class words, the per-word distribution of possible tags is usually sharply peaked around a few tags. This means that in most situations, there are only a few possible tags. Finally, the context required to disambiguate most words is usually only a few words and/or surrounding POS tags. This information is local. In turn this means that we are likely to see again some previously observed context and so know which decision to make. For some

non-English languages, current taggers are effective, but it is an open question whether they are equally effective for all languages.

1.7.2 Statistical Parsing

After the success of assigning syntactic categories to words, one might be interested in seeing whether broad-coverage parsing could be improved using statistical means. This task is considerably harder than tagging as we cannot be sure if the grammar actually parses most naturally occurring sentences; even if we could parse all sentences, we still need to select among potentially thousands of alternatives. Finally, we would like to make this selection efficient.

Within rule-based natural language processing, it is probably true to say that none of these problems were ever truly solved. For example, the Alvey Natural Language Tools Grammar can only parse around 20% of the Wall Street Journal (Grover et al., 1993). These parsers, while efficient, could never parse in-coverage sentences that had more than about 20 tokens (Carroll, 1993). More recent developments within rule-based parsing have increased the efficiency of the parsers, but the lengths of the sentences that they can effectively handle are still fairly short (Kiefer et al., 1999).

In contrast to these rule-based 'deep' parsers, there has been an alternative vein of work developing much shallower probabilistic parsers (Magerman and Weir, 1992, Collins, 1996, Charniak, 1999, Collins, 2000). These parsers usually contain grammars that are induced from parsed treebanks (such as the Penn Treebank). Furthermore, they usually also contain a probabilistic component which enables them to find the best parse for a given sentence, in near-linear time. Such systems are easily capable of parsing almost all naturally occurring sentences.

Finally, there has been some work on combining stochastic methods with more linguistically motivated parsers. For example, Johnson et al. (1999) showed how a broad-coverage LFG parser could use maximum entropy techniques to select the best parse. Osborne (2000) also showed how a DCG parser could also use similar techniques.

1.7.3 Text Classification

Moving up to documents, a useful task is *text classification*. This means assigning some text (which may be a book, an e-mail, a computer program, etc.) a label. The label will have a meaning depending on the domain. For example, in electronic commerce, the labels may deal with ordering products, reviewing the status of orders, and so on; in authorship determination, the labels are authors.

Text classification turns out to be a relatively easy task. The usual

approach is to reduce a text to a bag-of-words (an order independent set of words) and then treat these words as a vector annotated with the label of the text that it represents. Finally, these labeled vectors can then be used to train any one of several machine learning classification algorithms (Yang and Liu, 1999). Treating a document as a bag-of-words clearly throws away a lot of linguistic information. However, this abstraction reduces the dimensionality of the problem (the number of features of the task that need to be tracked). Minimizing the number of dimensions make training more efficient and also can improve performance (this relates to a problem called *overfitting*, which we discuss in Section 1.3).

1.7.4 Question Answering

The task of *question answering* moves beyond simple document classification into the realm of intelligent information retrieval. Question answering (QA) is concerned with retrieving short passages within documents that answer a user's question. QA is different from simple web searching because fully-formed questions encode information about the type of answer being looked for, where as keyword queries do not. For example the questions:

(a)Who was elected the 16th president of the United States?

(b)When was the 16th president of the United States elected?

indicate that the user is either looking for a name or a date, whereas the keyword query *16th, President, "United States", elected* does not. The QA process usually involves several different stages: first the question is analyzed for answer type, then candidate documents are marked up for named entities that correspond to that answer type, and finally short passages are selected and ranked by their probability of answering the question. All of these tasks can be performed within a SNLP framework. For example, Ittycheriah et al. (2000) used a maximum entropy model component to perform answer type classification. Leidner and Callison-Burch (2003) propose that answer ranking and evaluation of QA systems could be done automatically using a large collection of frequently asked question (FAQ) pages gathered from the web. These pages provide a potentially valuable source of information for a statistical question answering system since they constitute a ready source of questions paired with their answers. Rather than having to hand-craft rules for a question answering system, the components of a system might be learned automatically from a corpus assembled from internet resources.

1.7.5 Machine Translation

Unsurprisingly, *machine translation*, which is arguably the hardest task in NLP, has a statistical formulation (Brown et al., 1993, Knight, 1999). Suppose we are translating between English and French. In statistical translation, we take the view that every English string, e, is a possible translation of a French string f. We assign to every pair of strings $< f, e >$ a probability $P(e|f)$, which we interpret as the probability that a translator, when presented with f, will produce e as its translation. Given a French string f, the job of the machine translation system is to find that English string $\hat{e}$, from all possible English strings for which $P(e|f)$ is greatest.

To estimate $P(e|f)$, Brown et al. (1993) take the view that translation is a process of string re-writing where each string in the source sentence can be expanded into zero or more words in the target language; each word translates into a particular target word; and the words translated into the target language are then rearranged to reflect the syntax of that language. While string re-writing may seem like a simplistic way of viewing translation, it does have the advantage that its probabilities can be estimated from available data, which takes the form of bilingual sentence-aligned corpora such as the Canadian Hansard.

Recent work in statistical machine translation has taken two directions. The first is to augment the Brown et al. (1993) translation model with more linguistically sophisticated information. For instance Yamada and Knight (2001) formulate a syntax based translation model and show that it produces better translation results between English and Japanese than the re-writing method does. The other vein of research has to do with coping with scarce linguistic resources. Since statistical machine translation is learned from parallel corpora it does not tend to work well for language pairs which do not have extensive parallel corpora. Al-Onaizan et al. (2000) examines how human beings cope with scarce translation resources by seeing how well they manage to translate a new language (Tetun) into English given only a few examples. Callison-Burch (2002) applies weakly supervised learning to machine translation as a way of bootstrapping parallel corpora for new language pairs, and shows that a bilingual corpus consisting of 200,000 machine translated sentences achieves the translation accuracy of a human translated corpus containing 20,000 sentence pairs.

1.8 Discussion

SNLP is an exciting field that has brought together linguists, statisticians and machine learning researchers. As a result of this synergy,

computational linguistics has been invigorated by a concentration upon data and quantitatively testable theories. This flow of ideas is not one way: the machine learning community draws upon the language domain as a source of large, hard problems worthy of tackling.

In large part, the success of the SNLP enterprise has been due to the availability of annotated data: the Penn treebank, for example, spawned a large body of research. Access to annotated material significantly simplifies the estimation task to such an extent that we now have useful, robust tools such as POS taggers and probabilistic parsers that are routinely used by the community. As useful as these tools are, it could be argued that they are all relatively simple. For example, progress in broad-coverage parsers for linguistically sophisticated formalisms such as HPSG has been much less dramatic; we still have no robust methods for assigning logical forms to utterances in discourse. One reason for the relative simplicity of our current breed of tools is that suitable annotated material in sufficient quantities simply does not exist. Note that even for those tools we do have, it could be argued that performance is not at a maximum: more mileage could be had if only there was more annotated training material. In the section on data, we touched upon weakly supervised approaches such as co-training and active learning. These methods directly address the question of creating more annotated training material. It is very likely that these areas will become of central concern to researchers in SNLP in the very near future.

Orthogonal to the question of annotated material, there is also a need for better models of how language actually works. For example, while we have reasonable ideas how language behaves overall, we do not have good probabilistic models of how discourse varies, nor do we really know how language changes over time, or across domains. That is, language is usually assumed to come from the same source: it is modeled as *stationary*. Moving over to *non-stationary* language models is beyond the current state-of-the-art and is likely to be so for a long time to come. Simply throwing ever larger quantities of training material at bad models will not make them correct (Curran and Osborne, 2002). Instead, we need to develop sophisticated models that can deal with change in language. It is here that linguists are likely to have a key role to play.

This chapter sketched the field of SNLP and touched upon the current state-of-the-art and some of the problems that need to be tackled before we can fully analyze naturally occurring language. Progress in the field comes from linguists, computer scientists, statisticians and machine learning researchers. The cross disciplinary nature of natural

language is exciting. Having a better understanding of the tools of the trade - statistics and precise models of language - will allow you to make progress in the field.

1.9 Further reading

There are two good textbooks that cover SNLP: Manning and Schütze's "Foundations of Statistical Natural Language Processing" is a comprehensive treatment of the field. Jurafsky and Martin's "Speech and Language Processing" covers natural language processing more broadly, but does give significant treatment of statistical methods, including those used in speech processing. Other textbooks that provide introductions to relevant disciplines, but do not focus on SNLP, are Tom Mitchell's "Machine Learning", and DeGroot and Schervish's "Probability and Statistics".

References

Abney, Steve. 1996. Statistical methods and linguistics. In J. Klavans and P. Resnik, eds., *The Balancing Act*. Cambridge, MA: MIT Press.

Abney, Steven P. 1997. Stochastic Attribute-Value Grammars. *Computational Linguistics* 23(4):597–618.

Al-Onaizan, Yaser, Jan Curin, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz Josef Och, David Prudy, Noah Smith, and David Yarowsky. 1999. Statistical machine translation. Final Report, JHU Summer Workshop.

Al-Onaizan, Yaser, Ulrich Germann, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Daniel Marcu, and Kenji Yamada. 2000. Translating with scarce resources. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Blum, Avrim and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the Workshop on Computational Learning Theory*. Morgan Kaufmann.

Brill, Eric and Jun Wu. 1998. Classifier combination for improved lexical disambiguation. In C. Boitet and P. Whitelock, eds., *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 191–195. San Francisco, California: Morgan Kaufmann Publishers.

Brown, Peter, Stephen Della Pietra, Vincent Della Pietra, and Robert Mercer. 1993. The mathematics of machine translation: Parameter estimation. *Compuatational Linguistics* 19(2):263–311.

Callison-Burch, Chris. 2002. *Co-training for Statistical Machine Translation*. Master's thesis, University of Edinburgh.

Carroll, John. 1993. *Practical Unification-based Parsing of Natural Language*. Ph.D. thesis, University of Cambridge.

Charniak, Eugene. 1999. A maximum-entropy-inspired parser. Tech. Rep. CS99-12, Department of Computer Science, Brown University.

Charniak, Eugene. 2001. Immediate-head parsing for language models. In *Meeting of the Association for Computational Linguistics*, pages 116–123.

Chelba, Ciprian and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In C. Boitet and P. Whitelock, eds., *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics*, pages 225–231. San Francisco, California: Morgan Kaufmann Publishers.

Chen, Stanley F. and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In A. Joshi and M. Palmer, eds., *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318. San Francisco: Morgan Kaufmann Publishers.

Chen, Stanley F. and Ronald Rosenfeld. 1999. A Gaussian prior for smoothing maximum entropy models. Tech. Rep. CMU-CS-99-108, Carnegie Mellon University.

Collins, Michael. 2000. Discriminative Reranking for Natural Language Parsing. In *ICML 2000*.

Collins, Michael and James Brooks. 1995. Prepositional attachment through a backed-off model. In D. Yarovsky and K. Church, eds., *Proceedings of the Third Workshop on Very Large Corpora*, pages 27–38. Somerset, New Jersey: Association for Computational Linguistics.

Collins, Michael and Yoram Singer. 1999. Unsupervised models for named entity classification. In *Proceedings of the 1999 Conference on Empirical Methods in Natural Language Processing*.

Collins, Michael John. 1996. A new statistical parser based on bigram lexical dependencies. In 34^{th} *Annual Meeting of the Association for Computational Linguistics*. University of California, Santa Cruz, California, USA.

Cucerzan, Silviu and David Yarowsky. 2002. Bootstrapping a multilingual part-of-speech tagger in one person-day. In *Proceedings of CoNLL-2002*, pages 132–138. Taipei, Taiwan.

Curran, James R. and Miles Osborne. 2002. A very very large corpus doesn't always yield reliable estimates. In *Proceedings of CoNLL-2002*, pages 126–131. Taipei, Taiwan.

DeGroot, Morris H. and Mark J. Schervish. 2002. *Probability and Statistics*. Addison Wesley.

Dietterich, Thomas G. 1998. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation* 10(7):1895–1923.

Dietterich, Thomas G. 2000. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning* 40(2):139–157.

Ghani, Rayid. 2000. Using error-correcting codes for text classification. In P. Langley, ed., *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 303–310. Stanford, US: Morgan Kaufmann Publishers, San Francisco, US.

Grover, Claire, Ted Briscoe, John Carroll, and Bran Boguraev. 1993. *The Alvey Natural Language Tools Grammar (4^{th} Release)*. Tech. rep., University of Cambridge Computer Laboratory.

Henderson, John C. 1998. Exploiting diversity for natural language processing. In *AAAI/IAAI*, page 1174.

Ittycheriah, Abraham, Martin Franz, Wei-Jing Zhu, and Adwait Ratnaparkhi. 2000. Ibm's statistical question answering system. In *9th Text REtrieval Conference*. Gaithersburg, MD.

Joachims, Thorsten. 1998. Text categorization with support vector machines: learning with many relevant features. In C. Nédellec and C. Rouveirol, eds., *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142. Chemnitz, DE: Springer Verlag, Heidelberg, DE.

Johnson, M. 1998. Optimality-theoretic lexical functional grammar. In *Proceedings of the 11th Annual CUNY Conference on Human Sentence Processing*.

Johnson, Mark. 2001. Joint and conditional estimation of tagging and parsing models. In *Meeting of the Association for Computational Linguistics*, pages 314–321.

Johnson, Mark, Stuart Geman, Stephen Cannon, Zhiyi Chi, and Stephan Riezler. 1999. Estimators for Stochastic "Unification-Based" Grammars. In 37^{th} *Annual Meeting of the ACL*.

Junqua, Jean-Claude and Gertjan van Noord, eds. 2001. *Robustness in Language and Speech Technology*, vol. 17 of *Text, Speech and Language Technology*. Kluwer Academic Publishers.

Jurafsky, Daniel and James H. Martin. 2000. *Speech and Language Processing*. New Jersey: Prentice Hall.

Kiefer, B., J. Krieger, H-U. and. Carroll, and R. Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 473–480.

Knight, Kevin. 1999. A statistical MT tutorial workbook. Prepared for the 1999 JHU Summer Workshop.

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

Leidner, Jochen L. and Chris Callison-Burch. 2003. Evaluating question answering systems using FAQ answer injection. In *6th Annual CLUK Research Colloquium*. Edinburgh.

Magerman, David and Carl Weir. 1992. Efficiency, Robustness and Accuracy in Picky Chart Parsing. In *Proceedings of the* 30^{th} *ACL, University of Delaware, Newark, Delaware*, pages 40–47.

Malouf, Robert. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of CoNLL-2002*, pages 49–55. Taipei, Taiwan.

Manning, Christopher D. and Hinrich Schutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcus, Mitchell P., Beatrice Santori, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19.

Mitchell, Tom M. 1997. *Machine Leaning*. New York: McGrawl-Hill.

Nigam, Kamal and Rayid Ghani. 2000. Understanding the behavior of co-training. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*.

Osborne, Miles. 2000. Estimation of Stochastic Attribute-Value Grammars using an Informative Sample. In *The* 18^{th} *International Conference on Computational Linguistics*. Saarbrücken.

Ratnaparkhi, A. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Resnik, Philip. 1999. Mining the web for bilingual text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*.

Salomon, Jesper, Simon King, and Miles Osborne. 2002. Framewise Phone Classification using Support Vector Machines. In *7th International Conference on Spoken Language Processing*. Denver, Colorado.

Samuelsson, Christer and Atro Voutilainen. 1997. Comparing a linguistic and a stochastic tagger. In P. R. Cohen and W. Wahlster, eds., *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 246–253. Somerset, New Jersey: Association for Computational Linguistics.

Steedman, Mark, Steven Baker, Stephen Clark, Jeremiah Crim, Julia Hockenmaier, Rebecca Hwa, Miles Osborne, Paul Ruhlen, and Anoop Sarkar. 2003. Bootstrapping Statistical Parsers from Small Datasets. In *Proceedings of the 10th European Chapter of the Association for Computational Linguistics*. Budapest.

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 26–33. Toulouse, France.

Yang, Yiming and Xin Liu. 1999. A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–49. Berkeley.

Yarowsky, David. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*.

February 12, 2003