Grammatical Inference
A Magical Mystery Tour

Jason L Hutchens

March 11, 1996

Abstract

How do children learn language? Is it already in their heads when they are born, or do they acquire language through experience? Is it possible to program a computer to learn a language using techniques which are possibly available to the human infant?

Grammatical Inference strives to automatically infer a grammar for a language from a subset of the strings in the language, and possibly also a set of strings not in the language. There are many useful applications which will benefit from a general Grammatical Inference Engine, such as speech and image recognition, natural language processing systems, and database query engines.

In this paper I give a general overview of the relatively new field of Grammatical Inference, starting from its application in a speech recognition system. I cover most of the commonly used techniques, and discuss some of the objections raised by linguists to these techniques. In a subsequent paper I will describe my project in detail and will cover the more philosophical aspects of this field, such as the human language acquisition debate.

These papers will hopefully serve to illustrate the power of probabilistic approaches to Linguists and Engineers alike.

1 Introduction

Grammatical Inference is the process of programming a computer to automatically infer a grammar for a language [13]. I am particularly interested in the inference of natural languages as this problem is the most challenging and has interesting applications.

My project is a little different from what normally goes on around CIIPS. Some have suggested that it is more related to Computational Linguistics and Computer Science than Engineering. But wait a minute—we all use syntactic pattern recognition techniques. The name alone is a dead give-away! We represent our data by describing it at a higher level,[1] and by doing this we are implicitly using a grammar.

One major application of Grammatical Inference is finding a language model for use in speech recognition, and I will describe that shortly. But it can also be applied to any application where a time series of symbols occur.[2] A grammar can be used for recognising valid sequences of symbols, replacing lost symbols, compressing the data sequence, generating valid strings of data, and so on.

The potential application in image processing, especially character recognition, is enormous. However, it is not obvious how to represent this kind of data as a time series, and so other techniques may have to be developed.[3]

I will begin this paper by presenting a speech recognition system and showing why a language model is needed. This will take us on a journey through Shannon's information theory and language modelling in general to arrive at IBM's method for Grammatical Inference. After demonstrating some improvements to IBM's system, I will discuss the objections raised to Grammatical Inference by linguists.

[1] By fitting Gaussians and upwriting, for example.

[2] The symbols themselves may be anything at all.

[3] For example, a two-dimensional grammar may be required. I am not concerned with this at the moment.

2 Speech Recognition

A speech recognition system is intended to transcribe human speech. That is, it takes a speech signal as input, and provides text as output. Hopefully it does this on continuous speech regardless of the speaker, while performing well in the presence of noise. Humans can *wreck a nice beach* exceedingly well[4] so why can't our computer?

2.1 A Speech Recognition System

A model of a simple speech recognition system is shown in Figure 1.

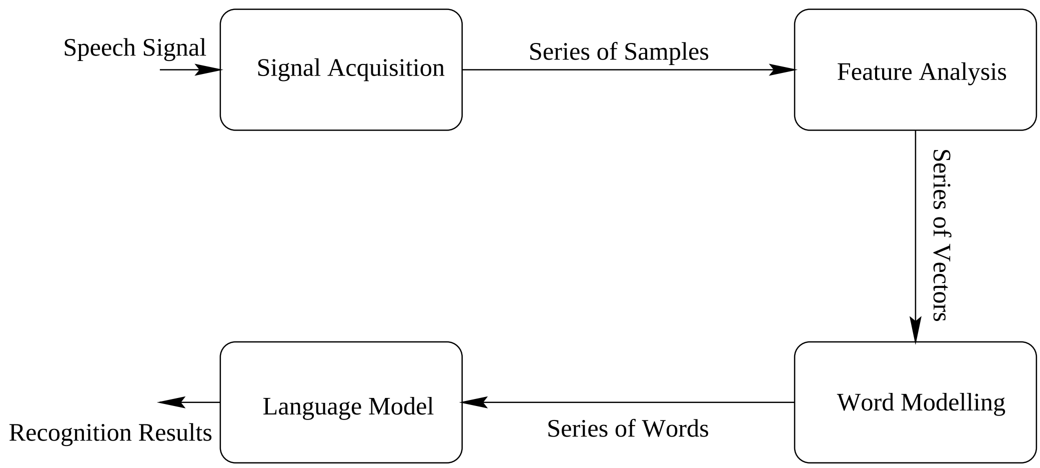

Figure 1: Model of a Speech Recognition System

The speech signal is typically taken from a microphone, and an Analog to Digital converter is used to turn it into something the computer can understand. This signal is then subjected to an analysis which gives a series of feature vectors. These vectors are then processed to give a model of phonemes

[4]Consider talking to an elderly relative over a bad telephone connection. Even though you miss some of what they say, you can make a good guess at what it was (not that you actually care.)

or words. Finally, a language model interacts with this acoustic model to give the actual recognition results.

The acoustic model may have trouble deciding which word was actually said, either because the speech signal was corrupted, or because several possible words could have caused the observed data. In such a case, it may give a guess in the form of a probability distribution over all possible words. As an example, consider the sentence below.

> "I'd like to take two cakes too!"

This is where a language model becomes useful. It uses the output of the acoustic model and the current context[5] to select the word that it deems most probable. In this way it can drastically improve the accuracy of speech recognition. For example, the IBM system can correctly transcribe the previous sentence when using a simple language model.

More formally, we wish our speech recognition system to select the string W which maximises $P(W|A)$, where $W = w_1, w_2, \ldots, w_n$ and A is the acoustic evidence. Using Baye's law, we get Equation 1. $P(A)$ is assumed to be constant, so the system must maximise $P(A|W)P(W)$ [11].

$$P(W|A) = \frac{P(A|W)P(W)}{P(A)} \tag{1}$$

The language model provides $P(W)$, which is usually given by Equation 2, while the acoustic model provides $P(A|W)$. We can use these two models in isolation to give the final recognition result. Because the language model must give $P(W)$, it is required to be a stochastic grammar. That is, it must be able to assign a probability to any string of symbols.

$$P(W) = P(w_1)P(w_2|w_1)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1}) \tag{2}$$

[5] The context is usually the string of words which have been recognized so far.

To date, most research in speech recognition has concentrated on the acoustic model.[6] In fact, it seems that a more significant improvement in speech recognition results can be obtained by working on the language model. In the last 30 years there has been a gradual increase of interest in this area, starting with the work done by IBM on their Tangora speech recognition system [1]. Only in the last 5 years have statistical text analysis and formal linguistic analysis been combined [3].

2.2 The Shannon Game

The task of the language model in a speech recognition system is one of prediction. Shannon encapsulated this idea in the Shannon Game, where the players are given a string of n symbols and must predict what the subsequent symbol will be [15]. For example, you may be presented with the text shown below.

> The cat sat on the

A stochastic grammar for the language would give its prediction in the form of a probability distribution, while a non-stochastic grammar would give a single symbol as its guess, as would a human being. If this were the case, the predictor would be told whether its prediction was correct or incorrect. If it was wrong, it could keep on guessing until it gets it right.

Now I am sure that the first word that popped into your head upon reading the sentence above was "mat". There are several reasons for this. Firstly, you have seen the sentence "The cat sat on the mat" many times before, and so you are expecting to see it again. Furthermore, your experience of cats has shown that they only sit on certain things, so even if you have never seen a sentence starting with anything like "The cat sat on the" before, you can still narrow down the possibilities.

[6] In fact, some researchers believe that that the language model is the easy bit, and it's the acoustic model which is a worry! They are, of course, mistaken in their belief.

In my role as the host of the game, I let you know that your prediction was incorrect. So perhaps you would then go on to predict words such as "chair", "floor" and "carpet" (because they are also things that cats sit on,) or "hat", "gnat" and "rat" (because they rhyme with "mat" and are also mildly amusing.)

This process may continue for several words, but soon you will become bored and demand that I tell you which word it actually was. Being a sympathetic person, I relent.

'The word was actually "ceiling" ', I say.

'WHAT?', you yell. 'Ceiling? What kind of madman are you?'

'It's a perfectly grammatical sentence.'

'Cats don't sit on ceilings—that's absurd! How do you expect me to play your stupid game if you pull silly tricks like that!'

And with that you throw up your arms in disgust and storm out of the room.

Such a demonstration shows that the grammar in your head is not like the formal grammars for pseudo-English developed by linguists.[7] It also indicates that the amount of shock you receive upon finding out the word is a crude measure of how well your grammar performed on my sentence. If the word "mat" actually did occur, it wouldn't have bothered you in the least.

2.3 Formal Language Theory

Before I go on to describe performance measures for predictors playing the Shannon Game, and Grammatical Inference techniques, I must give a brief

[7] As such a grammar can only predict that a noun-phrase is coming up, which could be anything from "mat" to "colourless green idea".

description of the terminology used in formal language theory. I have purposfully italicized the terminology the first time it is used to minimise confusion.

An *alphabet* is defined to be a finite set of fundamental units called *symbols*, denoted by Σ. For example, $\Sigma = \{a, b, x\}$ represents an alphabet of three symbols—a, b and c. The size of the alphabet is denoted by $\|\Sigma\|$, and in this case $\|\Sigma\| = 3$.

In English we usually refer to the alphabet as the vocabulary, and the symbols are the words you find in the dictionary. Actually, English has several levels of representation, so we have the alphabet of letters at a lower level.

The *language* is a set of finite string of symbols from the alphabet, and is denoted by the L symbol. Basically, $L \subseteq \Sigma^*$. Although the strings are finite in length, the set L may be finite or infinite. This set is specified by a *grammar* for the language.

In English strings of words are sentences. At a lower level, strings of letters are words. It is a bit confusing I know, but this concept will become clear in my second paper.

Hopefully the grammar generalises the language to some extent—that is, it compresses the language by representing all of the strings in the language more efficiently. If the language is finite, the set L itself is a valid grammar, but we can hopefully do a lot better than this. In this way, a list of words in the dictionary is a grammar for English on the letter level.

A string which is in the language is a *valid string*, while an *invalid string* is a string which is not in the language. A *recognition grammar* can test the validity of a string, while a *generation grammar* may generate valid strings. A *stochastic grammar* can not only do this, but can give a probability for the string as well.

2.4 Information Theory

Now that we know the fundamental terminology, we can move on to the interesting stuff. Our first problem is this—how do we compare two different language models? If we play the Shannon Game with two different predictors, how do we decide which one wins the game?

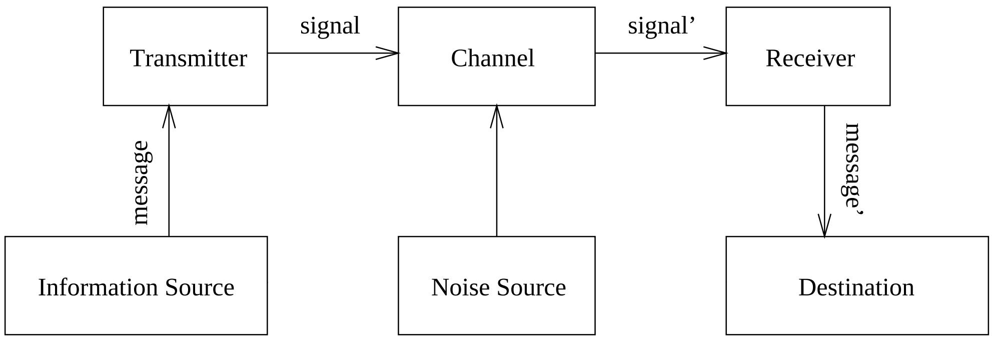

Figure 2: Model of a noisy communications system

Information Theory is concerned with transmitting a message[8] through a noisy communication medium as efficiently as possible. Redundant symbols may be added to the message to detect and correct errors caused by the noise, while redundancy can be removed from the message to compress it, and therefore transmit it more efficiently.

The English language has a high level of redundancy—it is this redundancy which helps us to understand what our elderly relatives are saying even when we don't hear some of the words. As an illustration, consider the text below.

> Cn u rd ths msg?

Even though this text is missing 10 symbols we can still understand it perfectly well. However, if it was corrupted by noise to give "Cx x rd xhs msx?"

[8] A message is simply a string of symbols.

we would have more trouble understanding it that if we were given a corrupted version of the original string, such as “Cax xou rexd xhis mxssaxe?”

Several measures were introduced by Shannon to quantify the information content of a received message, and the uncertainty that the receiver has about which symbol it is going to see next [16]. These measures have been adopted for use in the Shannon Game.

2.4.1 Surprise

The surprise experienced by the predictor when it discovers which word actually occurred is given in Equation 3, where $P(w|W)$ is the probability assigned by the predictor to word w, given that the string W or $n-1$ words had previously occurred. This equation is usually used in the form given by Equation 4.

$$I = \log_2 \frac{1}{P(w|W)} \tag{3}$$

This quantity is called the *information* provided to the predictor by the text, but I call it the surprise as it sounds more intuitive. In an information theory context, the information is the number of bits the symbol would require to be encoded in. That is, less probable symbols require more bits to transmit. This makes sense, as it will minimise the length of the entire string.

$$I = -\log_2 P(w|W) \tag{4}$$

If the predictor was certain w would occur, and it did, then the information recieved is $-\log_2 1 = 0$. Likewise, if the predictor was certain w would not occur, and it did, then the recieved information is $-\log_2 0 = \infty$. This case of the predictor being *infinitely surprised* should be avoided, and this is usually achieved by *smoothing* the prediction so that all words are assigned a non-zero probability.

The information supplied to the predictor by a string of words is equal to the sum of the information provided by the individual words. This is derived from simple laws of logarithms from Equation 2 to give the result in Equation 5.

$$I_{total} = -\sum_{i=1}^{n} \log_2 p(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{5}$$

We normalise the total information to provide a performance measure which isn't dependant on the size of the text, as given in Equation 6. This measure is the one commonly used for comparing predictors. The predictor with the lowest $\overline{I}$ is the winner of the game.

$$\overline{I} = -\frac{1}{n}\sum_{i=1}^{n} \log_2 p(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{6}$$

The maximum information is given in Equation 7, and provides an upper limit for the average surprise. It occurs when all words are equi-probable. It is useful for comparing how much better the predictor does than one which makes its predictions at random, without even looking at the words which have previously occurred.

$$\overline{I}_{max} = \log_2 \|\Sigma\| \tag{7}$$

2.4.2 Perplexity

IBM proposed the *perplexity* measure [10], as given in Equation 8. The perplexity is a monotone function of the average information, so it can be used as a measure of performance equally well as $\overline{I}$. This relationship is given in Equation 9.

$$PP = P(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}} \tag{8}$$

The perplexity gives the average number of equi-probable symbols that the predictor must choose between. That is, a language with PP equi-probable symbols would have the same average information. If $PP = \|\Sigma\|$, then you are in trouble! The perplexity of the model is used when evaluating the percentage improvement of one model over another.

Note that the information measures were in bits-per-symbol, while the perplexity is in symbols themselves. So we say that a predictor experiences an average surprise of x bits-per-symbol, and has a perplexity of y symbols.

$$PP = 2^{\overline{I}} \tag{9}$$

Because the perplexity gives the average number of words that the predictor must choose between, we can calculate the average number of guesses the predictor needs to get the word right. This measure, called the *rank*, is given in Equation 10.

$$R = \frac{PP + 1}{2} \tag{10}$$

The rank may be used to compare human predictors with computer predictors, as we can easily calculate the average number of guesses the human needs.

The *entropy* of a predictor is the surprise that it expects to receive when it finds out what the next word actually was. It is basically the amount of uncertainty the predictor has about what's coming up. The entropy is defined in Equation 11.

$$H = -\sum_{w \in \Sigma} p(w) \log_2 p(w) \tag{11}$$

There is also a method for calculating a performance measure for any grammar which is keeps guessing the word until it gets it right. This entropy is defined in Equation 12, where P_i is the probability that the predictor needs i guesses. P_i is estimated from counting the number of times the predictor took i guesses to get the word right, divided by the number of times the game was played [4].

$$H^+ = -\sum_{i=1}^{||\Sigma||} P_i \log_2 P_i \tag{12}$$

Note that H^+ is an upper-bound of $\overline{I}$. It can therefore be used as a crude performance measure between human beings and computers [11].

3 Grammatical Inference

Now we are ready to infer our language model. Keep in mind that although we will initially be using it to play the Shannon Game, the model will have other applications. The most obvious way of getting a model is to ask a linguist to design one for us. This approach has several disadvantages, though.

- The linguist designs grammars by working with a native speaker of the language—usually the linguist himself! But in many applications we only have a sample of text from the language, and no linguist can design a good grammar from this alone without assistance from a computer.
- If the model is to be used as a predictor in a speech recognition system, the grammar must be stochastic. A human being would find it impossible to design such a grammar.

- There aren't any useful grammars for natural languages at the moment anyway, so how can we expect to get one?
- If a grammar is found, it will very likely be quite rigid, and cover only a small subset of the language. As human beings frequently violate grammatical constraints when talking, this is innappropriate.
- We would ideally like the grammar to be *adaptive*. That is, the grammar should change when it encounters new material. We cannot hire a linguist to do this for us!

The other solution is of course to use a computer. At best we will have a large sample of strings in the language that we wish to infer a grammar for. As we are making no assumptions about this language, all we have is a sequence of symbols.[9] How does a computer get a grammar for that?

Well, recall Equation 2. All we need to do is estimate the probabilities $P(w_i|w_1, w_2, \ldots, w_{i-1})$ from the text, and this can be done using a straightforward statistical analysis. Such an analysis has many desirable properties which the linguistic approach lacks, such as the ones proposed by Charniak [3].

- The analysis is grounded in real data.
- It aims at problems where lack of perfection can be overlooked.
- Promises at producing usable results, and quickly.
- An obvious way to approach learning.

One major problem is that Gold proved that even the simplest of grammars cannot be inferred from a positive set of data only [9]. We will usually

[9]Actually, our assumptions are that the sample text is representative of the language, and that the computer knows how to read symbols in sequence from the text. That's all.

only have positive data though, as the existence of negative data implies the existence of an oracle. This shouldn't worry us too much, as we are interested in finding a grammar which performs reasonably well, and not the perfect grammar.

Fu proposed the structure of a Grammatical Inference Engine as given in Figure 3[7, 8], where $S+$ is the set of positive samples and $S-$ is the set of negative samples. Note that $S+ \subseteq L$ and $S- \subseteq \overline{L}$.

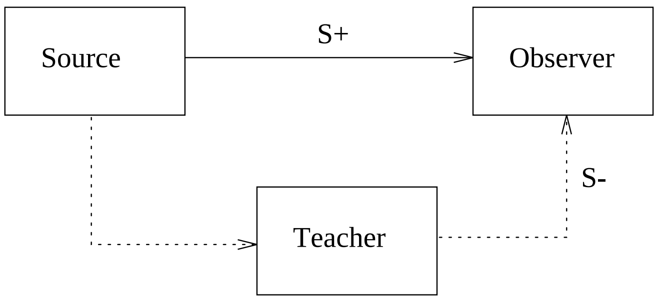

Figure 3: Structure of a grammatical inference engine

3.1 IBM Model

Our statistical analysis of the training text is required to estimate the probability $P(w_i|w_1, w_2, \ldots, w_{i-1})$. This can be done by simply counting the number of times the word w_i appears after the string $w_1, w_2, \ldots, w_{i-1}$, and normalising to get the frequency of occurence. That is, the probability of the word w_i is conditional on all preceding words.

However, this is way too much information for our language model to incorporate, so the past must be compressed into maneagable pieces. Strings of preceding words may therefore be grouped into *equivalence classes* prior to performing the analysis. For example, "The cat sat on the" and "The dog sat on the" may be in the same equivalence class.

The method used at IBM assumes that strings are in the same equivalence class if they end in the same two words [12]. This is of course false, but it simplifies things. For instance, all strings ending with the word sequence "on the" are considered to be identical. The assumption made by IBM is given in Equation 13.

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1}) \tag{13}$$

This language model is known as the *trigram* model, and it is basically a Markov chain with each state containing two symbols. In fact, Shannon proposed this kind of language modelling, and he used a *bigram* model to generate the text shown in Figure 3.1.

THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED

Figure 4: Shannon's 2^{nd}-order approximation to English

As you can see, this produces a pseudo-English text where the local dependencies are quite realistic, but long-distance dependencies are not. This is due to the nature of the model, in that each word is constrained by a context consisting of the preceding word only.

I will now proceed to describe the method IBM use to perform Grammatical Inference.

3.1.1 Statistical Analysis

Firstly statistical analysis is performed on the training text. This process simply runs through the text sequentially and retrieves all of the trigrams.

Once this process has finished, the trigrams are grouped according to whether their first two words are the same. For each trigram group a frequency vector is found, and this is used as an approximation to the probability distribition.

For example, if the text were `aababaaabba`, the trigrams would be `aab`, `aba`, `bab`, `aba`, `baa`, `aaa`, `aab`, `abb` and `bba`. The corresponding equivalence classes are `aa`, `ab`, `ba` and `bb`, and the four frequency vectors are $[\frac{1}{3}, \frac{2}{3}]$, $[\frac{2}{3}, \frac{1}{3}]$, $[\frac{1}{2}, \frac{1}{2}]$, and $[1, 0]$

The main problem that IBM found was that many possible English trigrams never actually take place in even very large text corpora. This means that if the predictor uses the trigram distributions alone, it will very often be infinitely surprised. A method is required to reduce this occurrence.

3.1.2 Smoothing

The idea that IBM hit upon was to smooth the prediction by making it a linear interpolation of the trigram, bigram and unigram distributions. That is, the probability distribution was estimated from the data as in Equation 14. Note that $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

$$p(w_3|w_1, w_2) = \lambda_1 f(w_3|w_1, w_2) + \lambda_2 f(w_3|w_2) + \lambda_3 f(w_3) \tag{14}$$

This reduced the occurence of infinite surprise significantly—in fact it will only occur if a new word is used as the interpolation process ensures that each word is assigned a non-zero probability of occurrence (provided $\lambda_3 \neq 0$.) In fact, this is the exact language model which IBM use in their Tangora speech recognition system.

3.1.3 Optimize

Our problem is now this: how do we find the best interpolation weights λ_1, λ_2 and λ_3? We must first do a bit of visualisation.

Consider a three-dimensional space with λ_1 and λ_2 along two dimensions, and PP on the third (λ_3 is redundant as it is not orthogonal to the other weights.) The possible values of λ_1 and λ_2 describe a triangular plane in R^3.

If we run the predictor for a particular value of the weights, and record the PP we get, we can plot the vector $[\lambda_1, \lambda_2, PP]$. If we do this for many different weights, we get a manifold. That is, our nice triangle will be all warped and wobbly.

The problem of finding the best interpolation weights is simply one of finding the lowest point on the manifold. But we'd like to do this without running the prediction algorithm too many times. To do this, IBM use the forward-backwards algorithm which is also known as the Baum-Welch algorithm.

To begin the forwards-backwards algorithm, we represent Equation 14 in the form of Equation 15, which may be used to form a Hidden Markov Model.[10] Note that the Φ_i's are equivalence functions which give the required valued from Equation 14.

$$\hat{P}(w_3|w_1, w_2) = \lambda_1 P(w_3|\Phi(w_1, w_2)) + \lambda_2 P(\Phi(w_3|w_2)) + \lambda_3(\Phi(P(w_3)) \quad (15)$$

Given a training sequence, the forward-backwards algorithm adjusts the weights to make the sequence as likely as possible. This algorithm is conceptually similar to that used to estimated the values of a normal Markov Chain, but more complicated in operation. The problem is that transition counts cannot be so readily obtained because there are many possible paths an HMM can take to produce a given output.

Luckily we can forget about which transition was actually used, and simply pretend that they were all used, but weight them by the probability of the

[10]A Hidden Markov Model is like a normal Markov Chain, except that there are many possible paths which give the same output.

transition. This requires knowledge of the probability, which is what we sre trying to find, so we must first estimate these probabilities and then iterate the algorithm, improving the estimate at each step.

Charniak lists some problems with this algorithm [3]. Basically, as with all optimization algorithms, the forward-backwards algorithm has several pitfalls.

- It only finds local minima.
- It cannot do better than training sequence allows, and therefore the solution may be unrepresentitive of the language.
- The solution is dependant on the initial estimate.

IBM have used the trigram predictor with the forward-backwards algorithm quite successfully in their speech recognition system.

3.1.4 Parts-of-speech

The IBM predictor is nice and elegant, but there is plenty of room for improvement. One saving grace is that extra terms may be added to the interpolation equation, and the forward-backwards algorithm will adjust the weights to favour the terms which have the greatest impact on the prediction results.

One problem with the IBM predictor is that if the trigram distribution is a zero-vector, then only the bigram and unigram distributions will be used. In order to get more contextual information to overcome this problem, a fourth term was added to the interpolation. This term is a trigram distribution generated from the parts-of-speech[11] in the training text [11], and the new

[11] Parts-of-speech are nothing more than nouns, verbs, adjectives and so on.

approximation is given in Equation 16. Note that g_1 and g_1 are the parts-of-speech of w_1 and w_2 respectively.

$$p(w_3|w_1, w_2) = \lambda_1 f(w_3|w_1, w_2) + \lambda_2 f(w_3|w_2) + \lambda_3 f(w_3) + \lambda_4 f(w_3|g_1, g_2) \quad (16)$$

The problem now is one of annotating the training text with the correct parts-of-speech. A linguist may come in useful here, but IBM developed the somewhat alliterative automatic annotation algorithm instead. It is beyond the scope of this paper to describe that algorithm here.

Using parts-of-speech in a language model violates my requirement that the Grammatical Inference Engine should possess no innate knowledge about the language. It would be better if the parts-of-speech were found automatically by the Inference Engine itself, and I will cover ways of doing just this in the next section.

After all this work, IBM found that the addition of the fourth term gave only a 5% improvement which wasn't significant enough for use. This was ascribed to the fact that the part-of-speech classification was too crude.

3.2 Chaters Model

Perhaps rather than using the parts-of-speech that linguists use, the Grammatical Inference Engine could infer them automatically from the training corpus. This is exactly what Chater has done [5].

Back in pre-Chomskian linguistics, words were assigned to syntactic categories using distributional approaches. That is, words were deemed to be similar if they could be used in similar contexts. Chater performed such a distributional analysis by forming a context vector for each word, which consisted of the previous and following two words. Clusters of these context vectors were found, and used as a basis for assigning words to syntactic categories.

The results are very impressive. Unfortunately the method used can only find one syntactic category for each word. This is clearly absurd for natural languages such as English, as a great proportion of the words are ambiguous. There has been some work into context-sensitive versions of Chater's algorithm, but it is still in its infancy [14].

3.3 Binary Decision Trees

Up until now we have been concerned with language models based on the trigram predictor. There are other ways of forming a language model, such as Binary Decision Trees [2]. In fact, the trigram predictor is a type of Binary Decision Tree.

Each node contains a question requiring a yes/no answer, (such as "is the previous word a verb?"), and each leaf contains a probability distribution which is conditional on the path of questions leading to that leaf. Hence the path is an equivalence class!

Bahl gives an algorithm for automatically growing the tree in a way that maximises the reduction in entropy contributed by each node added to the tree. Such a language model tends to capture long term dependencies, which the trigram model is unable to do.

4 Arguments Against Inference

Language modelling is the task of the linguist. Before the Chomskian revolution, linguists applied a distributional approach that used the contextual constraints of words as a basis for categorising them.

Traditionally, when an application has required a language mdoel of some sort, it has been designed by hand rather than inferred, as it is usually found to give a more accurate model.

Linguists have postulated several arguments against statistical approaches to language processing [6].

- Nonsensical strings and ungrammatical strings may be rejected by the language model on the same grounds, since they are both equally remote. Syntax and sense should be independant according to linguists, as Chomsky demonstrated with the syntactically correct but non-sensical sentence "Colourless green ideas sleeped furiously."
- Cannot capture long-distance word dependencies such as the ones exhibited in the sentence "The man who used to live down the street from us for about five years shot himself yesterday afternoon." In this sentence, the prediction of the seventeenth word 'himself' depends on the second word 'man'.
- Human beings cannot give you probabilities, so why should a computer program? Charniak does point out that this objection is false, as human beings can give a rough guess of the releative likelihoods of words [3].
- The probability of a word depends only on those before it! This objection is merely a misinterpretation of Equation 2.
- Statistical models are crude word-counting affairs. This objection is caused from confusing statistical models with crude statistical methods. The former do not require the latter [3].

Many prejudices against statistical models are really prejudices against unsophisticated statistical methods [3].

5 Conclusion

In this paper I have given a quick descripiton of Grammatical Inference. I have covered the most commonly used language model, which is the IBM trigram model. I have discussed bothe the advantages and disadvantages of Grammatical Inference, and the objections raised by linguists to such statistical models of language.

In a following paper I will introduce the work that I have done on Grammatical Inference. I plan to discuss the biological aspects of language acquisition, and to speculate on how the human infanmt acquires language.

References

[1] A. Averbuch, L. Bahl, R. Bakis, P. Brown, G. Daggett, S. Das, K. Davies, S. Degennaro, P. de Souza, E. Epstein, D. Fraleigh, F. Jelinek, B. Lewis, R. Mercer, J. Moorhead, A. Nadas, D. Nahamoo, M. Picheny, G. Schichman, P. Spinelli, D. Van Compernolle, and H. Wilkens. Experiments with the Tangora 20,000 word speech recognizer. Technical report, IBM T.J. Watson Research Center, 0000.

[2] L. Bahl, P. Brown, P. de Souza, and R. Mercer. A tree-based statistical language model for natural language speech recognition. Technical report, IBM T.J. Watson Research Center, September 1987.

[3] Eugene Charniak. *Statistical Language Learning*. MIT Press, 1993.

[4] Nick Chater and Martin Redington Stephen Finch. The guessing game: A paradigm for artificial grammar learning. Available at ftp://oliphant.cogsci.edinburgh.ac.uk/pub/frankred.

[5] Nick Chater and Martin Redington Stephen Finch. The potential contribution of distributional information to early syntactic category acquisition. Available at ftp://oliphant.cogsci.edinburgh.ac.uk/pub/frankred.

[6] Noam Chomsky. *Syntactic Structures*. Mouton, 1975.

[7] King-Sun Fu and Taylor L. Booth. Grammatical inference: Introduction and survey - part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3), May 1986.

[8] King-Sun Fu and Taylor L. Booth. Grammatical inference: Introduction and survey - part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3), May 1986.

[9] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[10] Frederick Jelinek. Markov source modeling of text generation. Technical report, IBM T.J. Watson Research Center, 0000.

[11] Frederick Jelinek. Self-organized language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center, 0000.

[12] Frederick Jelinek. Computation of the probability of initial substring generation by stochastic context free grammars. Technical report, IBM T.J. Watson Research Center, April 1990.

[13] S.M. Lucas. New directions in grammatical inference. In *IEE Colloquim on Grammatical Inference : Theory, Applications and Alternatives*, pages 1/1–7, April 1993.

[14] Hinrich Schütze. Part-of-speech induction from scratch. Available at ftp://csli.stanford.edu/pub/prosit/papers.

[15] C.E. Shannon. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:50–64, 1951.

[16] Claude E. Shannon and Warren Weaver. *The Mathematical theory of Communication*. University of Illinois Press, 1949.