VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences

Bigram Model Generalisation Using Singular Value Decomposition

by

Ben Wong

A thesis

submitted to the Victoria University of Wellington
in fulfilment of the requirements
for the degree of
Master of Science
in Computer Science

Victoria University of Wellington

October 19, 1999

Abstract

This thesis is about building good models for predicting sequences, in particular, the sequence of words contained in written English. It presents a technique for making a bigram model more general by using Singular Value Decomposition (SVD). The thesis describes a system that was implemented to evaluate the technique on bigram models that predict the sequences of words in English text. Experiments carried out with the system showed the technique to be effective at generalising bigram models that predict simple artificial sequences. However, experiments on real text suggest that the technique is not effective on bigram models that predict the sequence of words contained in real English text.

Acknowledgements

Firstly I would like to thank my first supervisor Tony Plate for his great ideas and patience with me; it is a wonder that he didn't throttle me at sometime during his stay here. Secondly, I would like to thank my second supervisor Peter Andreae for his encouragement and patience also, it was not easy teaching me how to write so that someone else could understand what I was on about.

I would like to thank all my friends, especially May-N, who at several times stopped me from quitting all-together. To the rest of you I give my thanks as well, without friends, this would be a really boring place.

I would like to thank my family, for the love encouragement, and food throughout the many, many years of my education.

I would like to thank the staff who have taught and dealt with me over my years here, and perhaps most recently Celia of the Science Faculty, who has dealt with me patiently and kindly, even at my most annoying.

And finally I would like to thank Aaron and Fiona, with whom I started this endeavour, but with whom I have finished without.

Contents

Chapter 1

Introduction

1.1 The Problem

This thesis is about using models to predict sequences of events, and improving those models to give better predictions. Prediction is a common task performed by people every day. From predicting the next colour of a traffic light, to picking a horse in the next race, prediction can be found in many common tasks and situations.

What many tasks of prediction have in common is that knowledge of the way things behave can help in predicting what happens next. For example: the next colour of the pair of traffic lights facing north and south can usually be predicted to be green after the west and east traffic lights turn yellow. The knowledge of the behaviour of a system is called a model. This thesis presents and tests a new method to improve models to give better predictions.

One useful modelling and prediction task that computers commonly perform is prediction of written or spoken English. By building a model of the English language, the predictions from that model can be used to compress text, recognise speech, and to help attribute authorship to works of dubious origins.

For this thesis, predicting the next word in a sequence of words contained in written English will be the basic task used to measure how well the new method performs.

Models of sequences of English words are prime candidate for improvement. The size of a model of a sequence of English words can be very large compared to the amount of information provided by even a large body of text. The statistics of a model built from a particular piece of text are therefore unreliable. It is the goal of the new method to improve models by reducing the effects of the lack of information.

1.2 The Approach

We plan to use bigram models of English words as a base model to be improved. Bigram models produce probability distributions that describe what words follow others. They do this by recording word pairs and producing the probability distributions from these pairs. Because there are many possible word pairs, it is always the case that many of the word pairs will be missing or rare — most word pairs appear only once or not at all, even in a very large body of text. This lack of information causes poor models to be built which give poor predictions of what word comes next.

Our new method is intended to leverage the similarity of components of the bigram model to produce a better model for the purposes of prediction. Many words have similar meanings or usages, and as such, we expect the distributions in which they are involved to be similar. Bigram models do not take advantage of this property and will contain redundant information by replicating the similar distributions or parts of distributions. The idea in this thesis is to exploit the redundancy in the bigram model and generalise the model so that the similar components inside the model are constructed from a set of higher level features which are derived from the model using Singular Value Decomposition (SVD).

1.3 Outline of this Thesis

Chapter 2 introduces a simple probabilistic model for predicting sequences of events called the bigram model. It also describes how to build bigram models and how to evaluate their performance on prediction tasks. Chapter 3 describes singular value decomposition (SVD), the core operation used in the method to improve bigram models.

Chapter 4 introduces a new method of generalising a bigram model by using SVD to approximate a bigram model encoded into a matrix. Chapter 5 describes a small exploratory experiment that tests out the new method on a small task of detecting the number of distributions in an artificially generated bigram model.

Chapter 6 describes a system to build, improve, and test bigram models of English text, and chapter 7 reports the results of how well the improved bigram models performed on several prediction tasks.

Chapter 2

Predicting Sequences with Bigrams

This thesis is about predicting discrete, nondeterministic sequences: primarily the sequences of words contained in written English. This chapter provides background on building and evaluating bigram models to predict such sequences.

Discrete sequences are sequences of symbols from a fixed set of possible symbols. An example of a discrete sequence is plain ASCII encoded text, which is a sequence of characters that come from a fixed set of possible letters and punctuation symbols. An example of a non-discrete sequence is a sequence of real numbers, such as the sequence of readings from a car's speedometer as it accelerates.

A nondeterministic sequence is a sequence that contains points where it is impossible to predict with certainty what the next symbol will be. An example of a nondeterministic sequence is the population size of an animal species over a number of years: it is impossible to predict exactly how many animals will die or be born in each year.

The sequences of words contained in written English are both nondeterministic and discrete. English text is discrete because the words contained in any given text are from a finite vocabulary or lexicon. English text is nondeterministic because at almost any given point in a piece of text, the author has a selection of words which could be used, and it is both what the author is trying to communicate and their personal preference which will govern their choice of words.

To predict a sequence is to determine by some means the next symbol in a sequence, usually by knowing the behaviour of the sequence. Because the sequences being dealt with in this thesis are nondeterministic, there is no way to always know exactly which symbol is coming next. The best prediction possible for a nondeterministic sequence is a description of the probabilities of each possible symbol being next, in other words, a probability distribution over all the possible symbols in the sequence.

2.1 Bigram Models

To make a good prediction of a sequence requires a model. A model of a sequence describes the sequence's behaviour and allows educated decisions or guesses on what symbols come next. This thesis uses *bigrams* to model the behaviour of sequences. A bigram is simply a sequence of two symbols, and can be thought of as an example or record of a transition between two symbols. By recording enough bigrams in a sequence, the long run behaviour of the transitions between symbols in the sequence is recorded. A record of every bigram contained in a sequence and the number of times they occurred is called a *bigram model*.

2.1.1 Building a bigram model

To build a bigram model of a sequence, all bigrams of that sequence are extracted, counted, and stored in a table. For example, the sequence

$$\langle a, a, b, a, b, a, b, a, a, b, a, b, a, b, b, a \rangle$$

has 15 bigrams, and would yield the model in figure 2.1.

Bigram	**Count**
$\langle a, a \rangle$	2
$\langle a, b \rangle$	6
$\langle b, a \rangle$	6
$\langle b, b \rangle$	1

Figure 2.1: A bigram model (M_1)

2.1.2 Using Bigrams to Predict Sequences

Bigram models record the behaviour of symbol transitions: that is, bigram models record how the sequence changes from one symbol to the next. Because bigrams can only express a transition from one symbol to the next, bigram models can only base predictions on one symbol, ignoring any other symbols which may, or may not be useful. For example, if a bigram model was used to predict the next symbol after seeing the sequence $\langle b, b, b, b, a \rangle$, it would only take into account the very last symbol (ie: a). The sequence of symbols seen up to the point which a prediction is made is known as the *context* of the prediction.

Bigram models make predictions of the form $P(\sigma|c)$, where c is the last symbol of the context and σ is a symbol for which we want to know the probability of being next. The general formula to derive a probability from a bigram model is:

$$P(\sigma|c, M) = \frac{\text{count}(\langle c, \sigma \rangle)}{\sum_{i=1}^{v} \text{count}(\langle c, A_i \rangle)} \tag{2.1}$$

Where M is the bigram model, and $\{A_1, \ldots, A_v\}$ is a set containing all the possible symbols, or *vocabulary*, of the sequence.

The bigram model M_1 shown in figure 2.1 would give the following prediction when using a as the context.

$$P(a|a, M_1) = \frac{\text{count}(\langle a, a\rangle)}{\text{count}(\langle a, a\rangle) + \text{count}(\langle a, b\rangle)} = 0.25$$

$$P(b|a, M_1) = \frac{\text{count}(\langle a, b\rangle)}{\text{count}(\langle a, a\rangle) + \text{count}(\langle a, b\rangle)} = 0.75$$

2.2 Evaluating Bigram Models

During the course of this thesis there will be a need to evaluate and compare the predictions that different models give. A prediction is a probability distribution over all the possible symbols describing the likelihood of each symbol being next in the sequence being predicted. A good prediction is a probability distribution that assigns high probability to the symbol that is next in the sequence.

We need a measure of how well predictions produced by a model fit a sequence, cross entropy is one such measure. For a bigram model M, and a test sequence $\langle s_1, \ldots, s_n\rangle$, the cross-entropy measure is:

$$\sum_{i=1}^{n} -\log(P(s_i|s_{i-1}, M)) \tag{2.2}$$

A good model will have a low value of cross-entropy. Note that this measure requires the value s_0, which does not exist. s_0 is required because bigram models are fixed context models, and they require exactly one symbol to make a prediction. The most common solutions for this is to use an additional symbol to represent the start of a sequence, or to use a separate model that does not need context to make the very first prediction.

The cross entropy measure is the sum of how well the predictions of the model fit the symbol transitions of the sequence. A poor fitting prediction is when a transition between two symbols in the sequence has a low probability in the model. For example, if model M assigns a low probability to $P(s_i|s_{i-1}, M)$, and the bigram $\langle s_{i-1}, s_i\rangle$ occurs in the test sequence, then $-\log(P(s_i|s_{i-1}, M))$ will be high, and the cross-entropy of M on that sequence will be greater because of the poor prediction.

For this thesis, all bigram models will be evaluated though their cross-entropy measure on some test set. By computing the cross entropy of many different models on the same test sequence, the relative performance of the different models can be measured.

The cross-entropy measure is closely related to the compression rate achievable by using the model and arithmetic encoding to encode the test sequence. Given two models and a sequence, the one with lower cross entropy will enable a greater compression rate on the sequence when the predictions generated by the model are used in conjunction with arithmetic encoding. For some comparisons (especially between models of different structure), we also

Sequence:

$$\langle a, a, b, a, b, a, b, a, a, b, a, b, a, b \rangle$$

Model (M_2):

	Bigram	**Count**
distribution 1:	$\langle a, a \rangle$	2
	$\langle a, b \rangle$	6
distribution 2:	$\langle b, a \rangle$	5
	$\langle b, b \rangle$	0

Figure 2.2: A sequence, and the bigram model of that sequence with a zero

need to take into account the complexity (or prior probability) of the models themselves. A less complex model, which has the same cross-entropy, is a superior model because it is more likely to capture the structure of the sequence.

The most notable comparison which requires a measure of model complexity is the task of transmitting a sequence over a network in the least number of bits as possible. This requires that the model needs to be transmitted along with the encoded sequence. For a transmission task on a sequence, the complexity of a model is embodied in the number of bits it takes to transmit the model.

2.3 Smoothing

Using just the frequency of bigrams to build a model does not often make for satisfactory predictions. Models based solely on bigram frequencies work only for some sequences, and only when there is enough data. If some bigrams are sufficiently improbable, or not enough data is available during construction of a model, then a bigram model may produce zero probabilities which are undesirable.

A bigram model can be considered to be a set of frequency distributions: where each distribution is made up of a sample of the bigrams on which the model is built. All bigram models can be split up into frequency distributions which describe behaviour after a symbol of context. For example, the model M_2 (shown in figure 2.2) has two distributions, one which describes the behaviour of the sequence after an 'a' occurs, and another which describes the behaviour after a 'b'. To split up a bigram model with an arbitrary number of symbols is trivial: each of the frequency distributions in the model is simply a grouping of all the bigrams which represent a transition from the same symbol (ie: all the bigrams in the training sequence that have the same first symbol).

Consider again the sequence and model in figure 2.2. Notice that the sequence and model have no record of the bigram $\langle b, b\rangle$. Model m_2 will produce a probability of 0/9 for a transition between two 'b's because no information about the transition has been seen. The occurrence of zero frequencies in the frequency distribution of a model is commonly known as "the zero frequency problem". Zero probabilities, and hence zero frequencies are undesirable because:

1. A probability of zero indicates that a transition is *impossible*, and will never happen in this kind of sequence. Such an assumption is unfounded when based on just lack of evidence.

2. If a probability of zero is given by a model, the cross-entropy measure of the model on any sequence which contains the unseen bigram becomes infinite. For a compression task using arithmetic encoding the number of bits used to encode the transition becomes infinite also.

The zero frequency problem is common for bigram models built with a word sequence from a body of English text. This is because most bodies of English text have a large vocabulary of words compared to the length of the text. A bigram model would require extremely large amount of text to gather statistics on even most of the common word bigrams, let alone the rare bigrams. The English language also has the property that most words do not follow most others, which leads to a large amount of bigrams which are never seen, no matter how much data is used.

The zero frequency problem is most commonly dealt with by either using more data, which is ineffective for models of word sequences, or by smoothing the distributions inside the model.

Smoothing is the action of "stealing" or redistributing frequencies from one part of a distribution to another. For the purposes of dealing with the zero frequency problem, the goal of smoothing is to get estimates of values for which the model does not have enough data to model reasonably.

The following subsections describe the methods used to smooth frequency distributions in this thesis.

2.3.1 Additive smoothing

Additive smoothing removes zero values by adding one to all values in the frequency distribution. Although additive smoothing seems like a terrible "hack", it is based on some quite reasonable statistics. Additive smoothing is equivalent to assuming a uniform distribution when there is no evidence to support otherwise. As more evidence or data is added to the model, the frequency distribution is deformed to follow more closely the distribution indicated by the data.

The most common criticism of the additive smoothing technique is that it takes no account of other knowledge that may be available, such as overall symbol probability. For example, in a bigram modelling a sequence of words from an English text, we know that the word "the" should be more likely than the word "isthmus", simply because it is used more often.

The additive smoothing technique is not restricted to always using one as the value to add. Values other than one can be used as the starting point for the frequencies in the distribution, this will change the importance of the initial uniform distribution depending on what value is chosen.

2.3.2 Good-Turing Re-estimation

The Good-Turing re-estimation method is based on an idea attributed to Alan Turing and refined by his associate I.J Good. The Good-Turing method takes an indirect approach by smoothing the "frequencies of frequencies" of a distribution, rather than the frequencies themselves.

For a frequency distribution which is a record of N bigrams, let n_r denote the number of times a frequency of r occurs in the distribution. ie:

$$\sum_{r=1}^{\infty} r n_r = N$$

Good-Turing re-estimation provides a re-estimation for values of r, known as r^*. r^* is r discounted by an amount based on the evidence of how many times the frequency r was seen compared to how many times the frequency $r+1$ was seen. The formula for r^* is:

$$r^* \approx \frac{(r+1)n_{r+1}}{n_r} \tag{2.3}$$

To compute a r^* value requires the values of n_r and n_{r+1}. For most frequency distributions this requirement is not fulfilled, especially for higher values of r, where many n_r values will not occur. Good-Turing re-estimation therefore first requires the n_r values to be smoothed to provide values for the missing values of n_r. Using smoothed n_r values, the formula now becomes:

$$r^* \approx \frac{(r+1)n'_{r+1}}{n'_r}$$

where n'_r is the smoothed value of n_r.

If the gradient of the n'_r values is negative as r rises, and its magnitude is decreasing as r rises, then the amount of discounting increases as r rises as well. Good-Turing will then steal more from the large frequencies, and less from the small.

The effect of discounting all the values this way means that the probability of all the classes represented in the sample is:

$$1 - \frac{n_1}{N}$$

And hence the probability of a new class is:

$$\frac{n_1}{N}$$

Good-Turing re-estimation assigns the same total probability to the set of all unseen bigrams as it does to the set of all unique bigrams (bigrams with a count of one). This allows the probability of unseen bigrams to be a proportion of the probability of unique bigrams, a reasonable assumption in many cases.

The Good-Turing smoothing scheme does not solve the additive smoothing problem of not taking advantage of extra knowledge. Good-Turing does however, provide a better way of finding the likelihood of seeing something new, rather than an arbitrary probability, such as done in additive smoothing.

Simple Good-Turing

Simple Good-Turing smoothing (SGT) is a popular algorithm for Good-Turing smoothing. SGT smoothes the n_r values to fit an exponential curve, resulting in a more mass begin taken from higher frequency classes than lower ones.

2.4 Other Models

Bigram models are simple and elegant, however, they have a few serious limitations. Bigram models are unable to cope with sequences whose behaviour is spread out over three or more symbols. For example, the sequence

$$\langle a, a, b, a, a, b, a, a, b, a, a, b, a, a, b, \ldots \rangle$$

requires that a model be able to look at two symbols of context to detect the obvious pattern that the sequence is following.

Bigram models are fixed context models, and must use exactly one symbol of context to make a prediction. For a sequence whose behaviour is not dependent on the symbol used as the context, the bigram model will waste effort modelling transitions which never improve the model.

2.4.1 Probabilistic models and n-gram models in general

Bigram models are part of a more general class of models known as n-gram models, which are in turn part of the class of probabilistic models.

Probabilistic models are the class of models that use probability distributions to represent behaviour of sequences. The general form of a probabilistic model is a state and a set of probability distributions that governs how the state changes.

n-gram models are a simple class of probabilistic models whose state is the last $n-1$ symbols of context, and whose probability distributions are determined by how many times that state has changed to each other state. n-gram models assume exactly n symbols of context are necessary to make predictions, just as bigram models assume one symbol of context is enough to make predictions. n-gram models use sub-sequences of length n to record the transition between the last $n-1$ symbols and the next.

n-gram models work in the same fashion as bigram models:

- n-grams are a sequences of n symbols and model transitions between the last $n-1$ symbols and the next; whereas bigrams are sequences of only two symbols and model transitions between the last symbol and the next.

- n-gram models are built by counting all n-grams which occur in a sequence; whereas bigram models are built by counting all bigrams which occur in a sequence.

To estimate the probability of a particular symbol coming next, we take the number of times there has been a transition from the current context to that symbol, and divide by the total number of times a transition has occurred from the current context to any symbol. The exact formula is:

$$P(x_m|\langle x_{m-n+1},\ldots,x_{m-1}\rangle) = \frac{\text{count}(\langle x_{m-n+1},\ldots,x_m\rangle)}{\sum_{i=1}^{v}\text{count}(\langle x_{m-n+1},\ldots,x_{m-1},\sigma_i\rangle)}$$

where $\{\sigma_1,\ldots,\sigma_v\}$ is the vocabulary of the sequence.

n-gram models that use longer n-grams are more expressive and can model more complex relationships. The drawback of using long n-grams is that the model becomes much more complex and requires more data to build. To gather information for all the n-grams requires extremely large amounts of data as the number of possible n-gram grows; for an n-gram model with v different symbols, there are v^n possible n-gram models to account for. Since the number of n-grams in a sequence does not increase as n increases, the exponential increase in model complexity causes the statistics to become less reliable. This is because the data is spread out over a larger number of n-grams and the amount of data available for each distribution is now smaller.

When there is not enough data for even a bigram model, a unigram model can be used. A unigram (which is another name for a 1-gram) is a sequence of one symbol. Unigram models only model the probabilities of symbols occuring, not the transition of symbols, but they concentrate the data into fewer distributions and therefore increase the reliability of the statistics.

All models which belong to the general class of probabilistic models can compute

$$P(X|\langle x_1,\ldots,x_{m-1}\rangle)$$

n-gram models approximate this by assuming only $n-1$ symbols of context are needed.

The cross-entropy measure can be computed for any probabilistic model using the formula:

$$\sum_{i=1}^{m} -\log(P(s_i|\langle s_1, \ldots, s_{m-1}\rangle, M))$$

Where M is the model begin evaluated and $\langle s_1, \ldots, s_m\rangle$ is the test sequence.

It is especially important to account for the complexity of a model when comparing between different kinds of model. A model which is an exact record of the test sequence will have a very low cross-entropy, but be very complex, while a bigram model, while not as accurate, will be much simpler.

2.4.2 Katz Back-off/Smoothing

Katz back-off is a method of combining n-gram models of different n into a single model. The Katz method uses Good-Turing re-estimation on the frequency distributions of the n-gram models, and then merges n-gram models with lower order models to introduce information from the lower order models to use their better statistics when appropriate.

2.4.3 Hidden Markov Models

Hidden Markov models (HMMs) are probabilistic models, much like n-gram models, but do not use the context as the state. HMMs are made up of a set of states, each of which has a probability distribution indicating the likelihood of a symbol being next, and a second probability distribution which indicates the likelihood of moving to the next state. In a HMM, the current state of the model represents a situation in the sequence and the behaviour of the sequence from that situation, rather than the behaviour of the sequence after a particular context has been seen.

Chapter 3

Singular Value Decomposition

This thesis explores the sequence modelling ability of singular value decomposition (SVD). SVD is an operation that decomposes any rectangular matrix into three component parts, which can be then joined back together to reconstruct the original matrix. By removing data from the component parts of the decomposition, an approximation of the original matrix is reconstructed: whose accuracy can be controlled by how much information was removed during reconstruction. SVD can also be considered a process which analyses a matrix into components of maximum variance: an equivalent process to principle component analysis. This chapter provides an overview of SVD and how it can be used to approximate matrices.

3.1 A Brief Definition of SVD

Singular Value Decomposition (SVD) is an operation which decomposes any rectangular matrix into three others, the product of which, reconstructs the original matrix perfectly, ie:

$$\begin{aligned} \mathrm{svd}(X) &= \{U, S, V\} \\ USV^T &= X \end{aligned}$$

Where:

- U and V are orthonormal[1]. The row vectors which make up U and V are known as the *singular vectors*.

- S is is a diagonal matrix with values sorted in descending order down the diagonal, ie:

$$S = \begin{pmatrix} s_1 & & \\ & \ddots & \\ & & s_n \end{pmatrix} \quad \text{where } s_i \leq s_{i+1}$$

  $s_1, \ldots, s_n$ are known as the *singular values.*

[1]Orthonormal means that all vectors which make up the matrix are at right angles to each other and have a norm of 1.

Some Notation

In this chapter there will be a lot of discussion about how singular vectors and values are used and interpreted with respect to the original matrix. To make explanation easier, the following notation will be used.

- X shall denote the original matrix to which SVD was done.
- U, S, and V shall denote the 3 matrices which form the decomposition of X.
- U_i shall denote the ith singular vector of U.
- V_j shall denote the jth singular vector of V.
- s_k shall denote the kth largest singular value in S (corresponds to S_{kk}).

3.1.1 Interpreting the Decomposition

After a matrix has been decomposed using SVD, the perfect reconstruction of a single cell of the matrix is given by the following formula:

$$X_{ij} = \sum_{k=1}^{n} U_{ik} s_k V_{jk} \qquad (3.1)$$ [2]

Where n is the number of singular values of S. Figure 3.1 shows how the reconstruction of X_{ij} can be interpreted as a scaled dot product between U_i and V_j scaled by S.

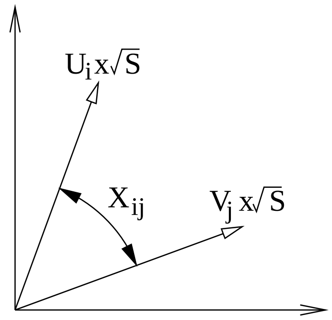

Figure 3.1: How X_{ij}, S, U_i, and V_j are related.

For any particular row in X, all cells are reconstructed using the same row of U and all rows of V. The singular vector U_i models the distinct features of the cells in row i of X. The singular vector V_j models the distinct features of the columns of X much like U_i models the rows of X: for any column of X, there is a single row vector of V which is used in the reconstruction of all cells in that column. U_i and V_j can be said to exist in a feature space, where the importance of each feature is specified by the singular values of S.

[2] $V^T_{kj} = V_{jk}$

3.1.2 Approximating Matrices Using SVD

One of the properties of S was that the singular values it contains are sorted down the diagonal of S. By setting the smallest singular values of S to zero, the least important features are dropped from the reconstruction, resulting in an approximation of the original matrix.

By only retaining the largest m singular values of S $(s_1, \ldots, s_m)$ and setting the rest to zero, the components of U_i and V_j which contribute least to the reconstruction have no effect on the reconstruction. We can re-multiply the reduced form of S with U and V together and get an approximation[3] of X, ie:

$$X \approx U\mathcal{S}V^T$$

Where $\mathcal{S}$ is the reduced form of S.

This property of extracting feature-space vectors is very powerful and is the basis behind other related work using SVD. Using this power directly is however, not what this thesis is about.

By removing singular values (setting them to zero) we in effect shorten the vectors of U and V. This shortening of the vectors which describe the matrix is known as dimensionality reduction, because the underlying model of the matrix is now expressed using vectors with less dimensions than before.

From a clustering/classification view, SVD is equivalent to doing a principle component analysis: where dimensions of most variance of data are extracted to find the most important components.

3.2 Related Work Using SVD

SVD is not a new technique, it is a well known method for solving or diagnosing systems of linear equations when other easier techniques fail. In fact, the use of SVD to model data is not new either, SVD is a well known technique for solving linear least squares problems.

What is new about SVD is the ability to find large numbers of singular values from very large matrices. The growth in computing speed and new algorithms to do SVD have made more applications possible away from the field of linear algebra.

The Latent Semantic Analysis (LSA) project [LD97] use SVD to build models of text whose purpose is to model the meaning of words. By deriving simple vector descriptions of words from the text, and then analysing these vectors with SVD, a model of what words mean is produced. By using the decomposition provided by SVD, the model can be used to compare how well words are related to each other, and to the text that they appear in.

Through the use of SVD, the LSA project has been able to mimic the word acquisition of children, something which has been impossible up to now. It is their hypothesis that some

[3] In fact a least squares approximation.

sort of dimensionality reduction is what humans do to extrapolate the meanings of words and are hence able to acquire words at a greater rate than which would be expected from learning the meanings of individual words one at a time.

Chapter 4

Combining Bigrams and SVD

One of the primary goals of this thesis is to generalise bigram models using SVD. This chapter brings together SVD and the bigram models of chapter 2, describing how to generalise or approximate a bigram model using SVD, and what it means to perform SVD on a bigram model.

4.1 Approximating Bigram Models Using SVD

Approximating a bigram model using SVD requires that the model be first translated into a matrix, then decomposed using SVD, and finally reconstructed using a lower number of dimensions.

4.1.1 Translating bigrams into matrices

The bigram model must be translated into a matrix because SVD is a matrix operation. Furthermore the translation from a bigram model to a matrix must preserve the structure in such a way as to keep relevant information and preferably, map logical structure in the model onto logical structures in the matrix.

The bigram model is obviously very similar in structure to a matrix and the translation is trivial: a bigram model is a set of probability distributions, one for each word of context, describing the likelihood of each symbol following the context. By representing the probability distributions as row vectors in a *bigram matrix* (see fig. 4.1) the probability distributions of a bigram model can be encoded without loss. There is no need in this case to encode the actual contexts or "next symbols" of the model other than to create a symbol table that enumerates each symbol so that they correspond to the indices of the bigram matrix.

4.1.2 Singular value decomposition on a bigram matrix

A bigram matrix is just like any other matrix and needs no special considerations to perform SVD on. But the decomposition resulting from the SVD on a bigram matrix of words is

context	next symbol a	b	c
a	1	2	3
b	9	8	7
c	5	5	5

$$\Rightarrow \quad \text{Bigram matrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 9 & 8 & 7 \\ 5 & 5 & 5 \end{pmatrix} \qquad \text{Symbol table} \quad \begin{bmatrix} a \to 1 \\ b \to 2 \\ c \to 3 \end{bmatrix}$$

Figure 4.1: Translating a bigram model into a *bigram matrix*

special. If the bigram matrix X was decomposed into U, S, and V where $X = USV^T$, then the row vector U_i is a singular vector representing the distinct properties of the context i, and V_j is a vector representing the properties symbol j as a following symbol.

By treating the vectors of V as properties of how symbols follow others, then the ith distribution (or row) of a bigram matrix is a linear combination of those properties. The role of U_i is to determine how important those properties are for ith distribution in the matrix, and the role of S is to determine how important each property is over the entire matrix.

By using SVD to approximate the matrix, the bigram model is now represented using less features: by dropping singular values the singular vectors are shortened and the least important features are dropped. The aim of this thesis is to find out if these smaller features are redundant in a bigram model. If so, removing them should improve the model so that it better models the true nature of the sequence.

4.1.3 The reconstruction and approximation of a bigram matrix

The bigram matrix may be reconstructed and approximated like any other matrix, but it may need additional processing afterwards. A bigram matrix which has been reconstructed may contain negative numbers or zeroes because the approximation done by SVD attains a mathematical approximation rather than a probabilistic one, and small values may be approximated to the point where they become less than or equal to zero. There are many methods for getting rid of zero probabilities as seen in chapter 2 but these will not deal with negative values sensibly.

Softmax and Exponential tail are two methods which overcome this problem.

Softmax

The softmax transformation stops negative values from occuring though performing SVD in log-space where all negative values represent positive values in normal space. To approximate the bigram matrix X, using this method requires 3 steps:

1. Transform to log-space by taking the natural logarithm of all values X to form $\mathcal{X}$.

2. Use SVD to approximate $\mathcal{X}$ to the desired dimensionality to form $\mathcal{Y}$.

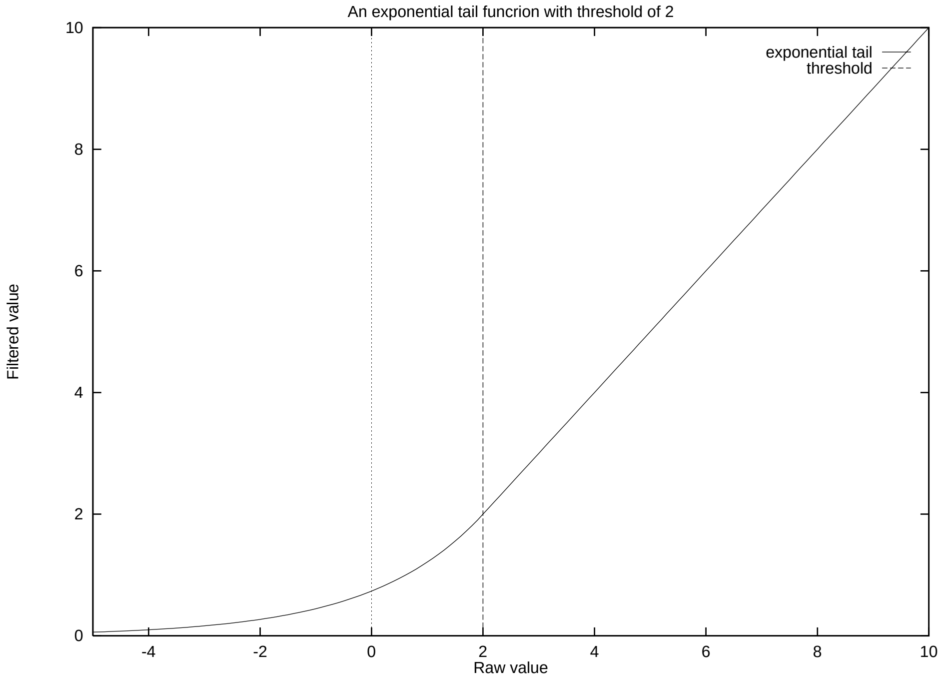

Figure 4.2: The exptail function with a threshold of 2

3. Restore $\mathcal{Y}$ to normal-space by taking to the power of e all values in $\mathcal{Y}$ to form the final approximation with no negative values.

This guarantees no negative values because any number transformed from log-space to normal space is positive; however the matrix approximated is no longer a bigram model, but rather a "squished" bigram model, and will alter the approximation produced at the end.

Exponential tail

Another method to deal with values which are too small or negative is to filter them through a function with retains the order of all values but transforms values below a certain threshold.

The exponential tail function (shown in figure 4.2) is a piecewise function, above the threshold it is a linear function, and below is a fitted exponential component. The family of functions used in this thesis is:

$$\text{exptail}(x,\theta) = \begin{cases} x, & x \geq \theta \\ ae^{bx}, & x < \theta \end{cases}$$
$$a = \frac{\theta}{e}; \qquad b = \frac{1}{\theta}; \qquad \theta \geq 0$$

Where θ is the threshold.

This family of functions ensures that:

1. as x tends to negative infinity, $\text{exptail}(x,\theta)$ tends to zero.

2. the slope of the tail must equal 1 at θ.

3. the value of the tail must equal θ then $x = \theta$.

4.2 Summary

This chapter has described a method of re-estimating the frequency distributions in a bigram model to make it more general. The method is based on the theory that approximating the representation of a bigram model with SVD will improve the estimates for a bigram model.

Chapter 5

An Exploratory Experiment

This chapter describes an experiment on artificial data which demonstrates some of the modelling capabilities of generalising bigram models using SVD. The aim was to find out whether it would be feasible to use SVD approximation to detect and model repeated distributions in a bigram model.

We expect bigram models of the word sequences contained in real text will have some repeated distributions or distributions which are similar to each other. The way that words are used has structure, a simple example being that nouns usually follow adjectives. It is the small structures and relations between different types and classes of words in real text which leads us to believe that a bigram model generated with real text will also have repeated distributions. The following experiment is an exploration to see whether SVD will be able to assist in making better models of real text.

5.1 Hypothesis

We hypothesise that a bigram model with repeated distributions is redundant and can be generalised using SVD. We also hypothesise that there will be a dramatic decrease in the quality of the model when the number of dimensions used to approximate the model falls below the number of different distributions.

5.2 Method

The following steps show the procedure by which the number of distributions were estimated using SVD.

1. We began by making an artificial bigram model that simulates a bigram model trained on a sequence with 10 different symbols.

   The first step was to choose N, the number of distinct distributions in the artificial model, where N was always between 1 and 10.

N distinct probability distributions were generated and assigned to the 10 true transition probability distributions of the simulated model. If N was less than 10, distributions are repeated in the model.

The transition probability distributions of a bigram model are a set of probabilities that each represent how likely a symbol is to come next. The distributions of the artificial bigram models were generated by sampling a stable distribution for each possible symbol and then normalising to produce a probability distribution. A stable distribution is one whose mass is distributed so that most of the mass is concentrated in low values, but a significant proportion of the mass is stored in a very long tail. This results in samples from the stable distribution that mostly have low values, but quite often, a very large value will be sampled. Making transition distributions this way creates distributions which have low probabilities for most transitions, but high probabilities for the remaining transitions.

The transition distributions were then sampled a total of 20,000 times to provide artificial bigram counts for the bigram model. Using 20,000 samples ensures a very good model is produced, with no zero values, and which is a good "base model" to evaluate approximated models against.

The counts were stored in a bigram matrix ready to be decomposed using SVD.

It should be noted that these are not bigram models of actual sequences. In particular a normal bigram matrix can be transposed to give a bigram matrix modelling the sequence in reverse; this is because the second symbol of any bigram (except the last one) from a sequence of symbols is the first symbol of the next bigram. With the generated models this is not the case.

2. SVD was performed on the artificial bigram matrix.

3. The matrix was reconstructed using 1 to 10 dimensions. To deal with zero and negative values, the exponential tail and softmax methods detailed in the previous chapter were used. For the exponential tail method, the minimum threshold was set to 1.

4. The quality of each of the reconstructions was then calculated. In this experiment the quality of the model is the cross-entropy of the reconstruction when tested on the bigrams (samples) contained in the original matrix. Because the bigram model consists of just frequencies of bigrams which were used to build the original model, it is easy to compute the cross-entropy of a bigram model on the raw data of another. The expression for the quality of a reconstruction is the sum of the encoding lengths of all the bigrams in the approximated model multiplied by the number of times they

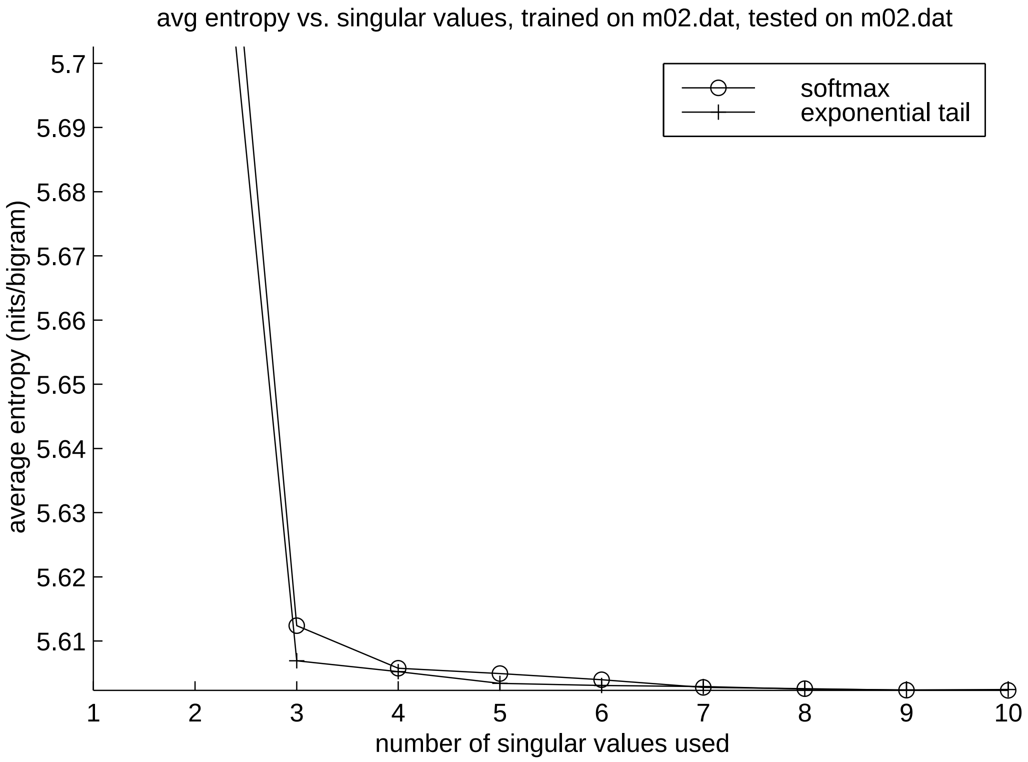

Figure 5.1: Example set of entropy values from a bigram with 3 distributions

occurred in the original matrix:

$$\text{cross-entropy} = \sum_{i=1}^{10}\left(\sum_{j=1}^{10}\left(-\log\left(\frac{Y_{ij}}{\sum_{k=1}^{10} Y_{ik}}\right) \times X_{ij}\right)\right)$$

where X is the original matrix and Y is the reconstructed matrix.

5. The cross-entropy values of the 10 reconstructions were then plotted onto a graph as in figure 5.1. The number of distributions in the data was then estimated by looking at the graph and identifying the point at which the cross-entropy increased dramatically. It should be noted that the estimation of the number of distributions and the generation of data was done by two different people to avoid bias during the estimation.

This process was repeated 30 times, each time selecting a value of N randomly. The difference between the estimated and the actual number of distinct distributions contained in the bigram model in each repetition were recorded and are summarised in figure 5.2.

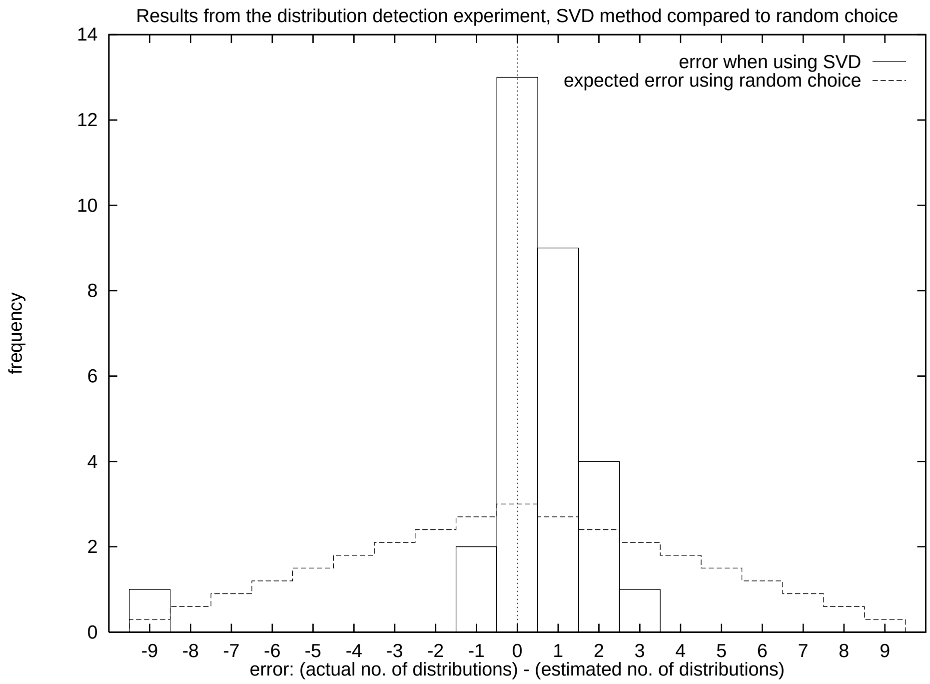

Figure 5.2: Results using SVD to detect the number of distributions

5.3 Results, Analysis and Conclusions

The results of this experiment were very encouraging, the estimates produced are far better than random guessing, with a slight tendency to over-estimate the number of distinct distributions.

The results shown in figure 5.2 show that:

1. Using SVD approximation makes it possible to detect the number of distributions from small sets of data.

2. Generalising bigram models using SVD has the potential to remove redundant information from a bigram model. The removal of dimensions from the U, S, and V matrices represents an actual loss of data with little reduction in performance, indicating the the extra dimensions were of little or no use.

3. In small bigram models the singular values produced by SVD are directly related to the number of distinct distributions inherent in the bigram model.

The small difference in figure 5.1 between using softmax and exponential tail methods, with exponential tail being slightly better than the softmax, was consistent throughout the entire experiment. This is likely to be because the exponential tail method did not deform the model in any way because of the high counts in the starting model. The softmax method however, must always deform the model, which is likely to be undesirable with such a good starting model.

5.4 Summary

On artificial data, generalising bigram models using SVD looks promising: SVD can be used to detect repeated distributions by modelling them using singular vectors and values. This ability also indicates a potential for modelling the distributions of words in a bigram matrix made up from real text, where certain sets of words are more likely to follow others, such as the set of words that commonly follow walk, run, and crawl.

Chapter 6

A System to Predict Sequences of Words

In the previous chapter, an experiment exploring the ability of SVD to generalise bigram matrices showed some promise. The experiment showed that it is possible to generalise artificial bigram models with repeated distributions to the point at which the number of distinct distributions can be detected using SVD.

Our premise is that the transitions between words contained in English text follow patterns of usage which are repeated or similar to each other. The reasoning is simple: words are the smallest element of speech which have *meaning*, and it is the meaning of a word which largely defines its usage. Our theory is that words of similar usage should be represented by similar distributions in a bigram model, and hence such a model can be improved using SVD.

This chapter presents a system that will build and generalise bigram models of English text to test whether they can be improved by approximating them using SVD.

6.1 System Overview

The system takes written English, in the form of plain text, as training data, and builds and evaluates generalised bigram models based on the sequence of words contained in that training data (see figure. 6.1).

The goal of the system is to build and test bigram models generalised using SVD. Figure 6.1 shows an outline of the system. The system takes two pieces of text as input: the first is a training set used to build and generalise a bigram model, and the second is the test set used to calculate the performance of the generalised models.

The system is split into three subsystems:

Modelling: The building of a bigram model of word sequences contained in raw text.

Generalisation: The building of generalised bigram models using SVD.

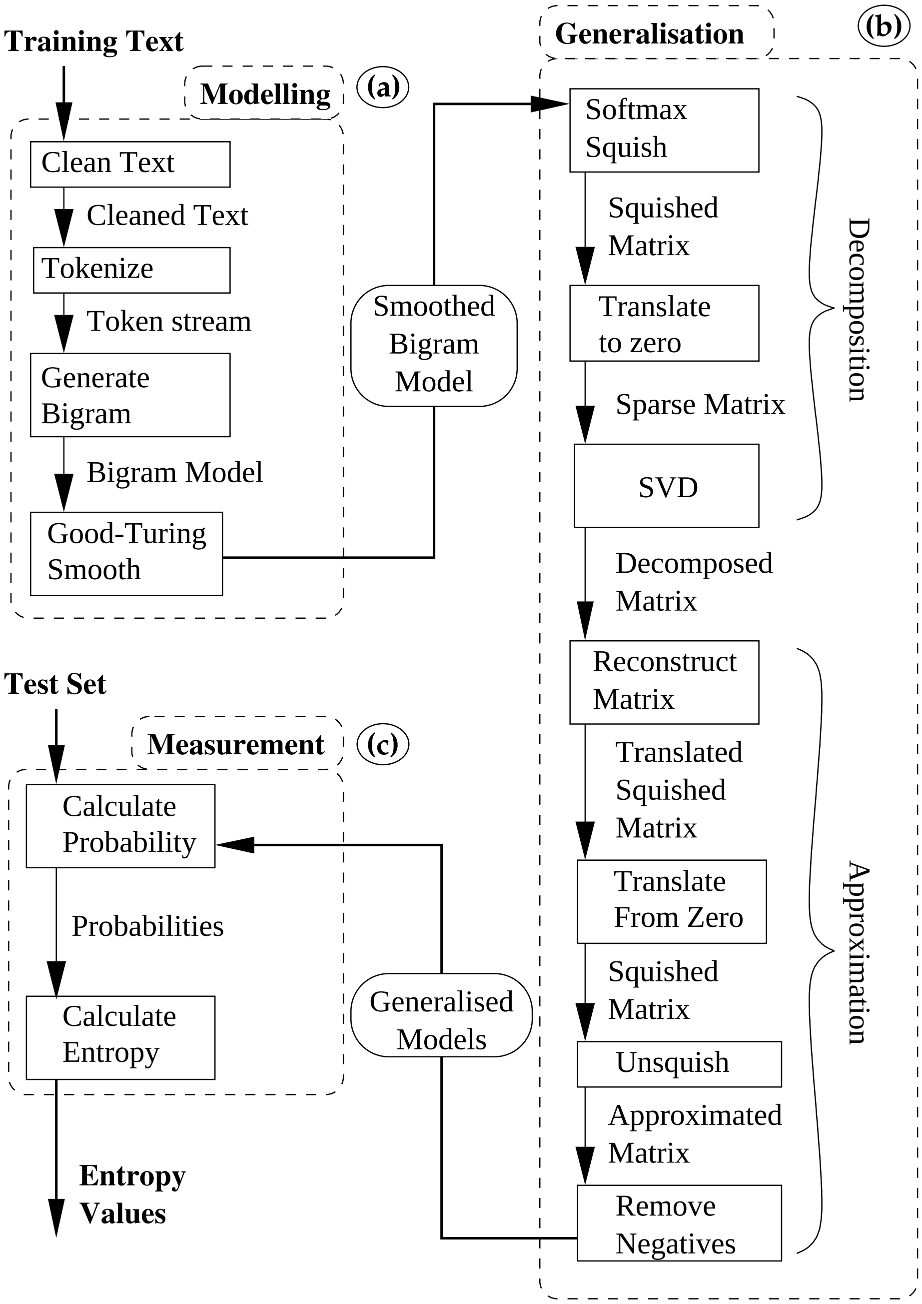

Figure 6.1: Logical view of the Program

Measurement: The evaluation of the generalised models.

The following sections describe the design of the three parts.

6.2 Modelling

This section describes the modelling subsystem shown in figure 6.1(a) which builds a bigram model of the word sequence contained in a piece of English text. The goal of this subsystem is to produce a bigram model with the very best distributions possible so that it can then be generalised using SVD.

6.2.1 Clean Text

Modelling sequences of words derived from written English text is difficult because of four reasons:

1. There are a lot of different words.
2. Typographical conventions and errors add noise to the sequence.
3. The way in which words are used and ordered is very complex.
4. English text is a "natural" dataset, different authors and styles of writing lead to different patterns of word usage.

These reasons lead to large and complex models that have heavy computational, memory, and training data requirements. In order to make modelling easier, the data needs to be cleaned. Cleaning data is the act of processing the data to make it less complex, or of higher quality. For example: removing capital letters or punctuation would simplify the data making it easier to model, and correcting spelling mistakes or adding words-tags (ie verb, noun, etc ...) would increase the quality of the data.

Cleaning text for modelling is a very specialised process and is often dependent on the body of text being cleaned as to what should be done. Because of this, no cleaning is done by the system itself, but rather done by using a variety of unix-based tools tailored to each case.

6.2.2 Tokenize

The plain text data that is given to the system is best described as a sequence of characters; the words contained in the text must be extracted and a stream of tokens produced. Each token in the stream represents a word or piece of punctuation, and it is these tokens that are used to generate the bigram model.

This system uses the same tokenizer used for the Penn Treebank Project, which separates all words and punctuation into distinct tokens, but leaves abbreviations as single words (eg “approx.” and “Mr.” are considered tokens), and splits contractions into two parts representing the original words, for example “won’t” becomes two tokens: “wo” and “n’t”.

The tokenizer requires that the text be first sentence tokenized, that is, all the individual sentences in the text must be identified. The system uses a rather ad-hoc method of sentence tokenization which simply assumes all full stops, question marks, and exclamation marks delimit sentences — except for those which occur within speech marks or are part of a known abbreviation (such as “Mrs.”).

6.2.3 Generate Bigram

Generating a bigram model from the token stream is the same as for any other sequence — count and record the number of times each bigram occurs in the sequence.

In a later stage of the system, singular value decomposition will be performed on the model. To make the model amenable to SVD, the bigram counts are represented as a bigram matrix as described in chapter 4.

Bigram models of word sequences from written English are very large: the models used in the system are expected to have around 10^5 different words and punctuation symbols. The data structure used to store the bigram matrix must therefore be given extra consideration to cope with this large number of possible tokens.

Dealing with large bigram models

The bigram models used in the system are of size proportional to the length of the text being modelled; making then very large and unwieldy. This is because the vocabulary of a piece of text with N words is typically $O(\sqrt{N})$ ([Tea98] page 73). Since the number of possible bigrams is the vocabulary squared, the number of possible bigrams to be modelled is $O(N)$. The bigram models of text used in this system are therefore potentially very large since datasets with millions of different words are common.

The large size makes representing the counts of all possible bigrams in the model undesirable. Because most possible bigrams do not occur in English text (most words do not follow most others), the data can be considered sparse. For example, the first 27 chapters of the novel “Emma” by Jane Austen’s has 7,875 different words. The number of possible bigrams which must be modelled is therefore $7,875^2$ or 62,015,625. Only 62,745 different bigrams (about 0.1%) occur; this very low proportion of non-zero bigram counts provide a solution to the problem of storing large bigram models. The solution is to have a sparse representation of the model, where only bigrams which occur are represented and all others are assumed to be zero. A sparse representation of the bigram matrix will allow great savings in memory consumption and disk space.

The sparse representation used in the system is the MATLAB sparse matrix format (see figure 6.2) which is a series of matrix index pairs followed by the value at that index.

$$\begin{pmatrix} 23 & 0 & 0 & 0 \\ 0 & 0 & 0 & 46 \\ 54 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ is stored as}$$

row	**column**	**value**
1	1	23
2	4	46
3	1	54

Figure 6.2: The MATLAB sparse matrix format

Modelling unseen words

The bigram model being built from the training set will eventually be tested using a separate test set. This test set may contain words not found in the training set and hence will not be represented when building a bigram model from just the training set. To model probabilities for the unseen words the test set is scanned for new words which are then included in the bigram model.

At first glance, the scanning of the test set during the building of the model indicates that the evaluation of the models will be biased. It is standard practice to have the test set unavailable during the building of a model to stop it from being a model of the test set rather than a general model. However, the only data given from the test set is the number of unseen words: no counts are given and no actual data about bigrams containing these unseen words is given. It is therefore reasonable to assume that the model will not be biased by the test data.

6.2.4 Good-Turing Smoothing

The intention of the modelling subsystem is to produce the best possible distributions in preparation for building generalised the models using SVD. Ideally these distributions should be close to the true distributions underlying the writing of the text, but the amount of data required to produce a good approximation is not obtainable. In fact, the true distributions underlying the usage of words in the text are an expression of the author's style and preferences, and as such they are constantly changing, meaning that there will never be enough data to produce counts for all the bigrams in the model. The lack of data means that the bigram model must be smoothed to provide better estimates so that the model to be approximated is as good as possible. Smoothing will generally replace the zeroes with non-zero values.

The sparse representation used to encode the model is now insufficient for the needs of the system: representing a smoothed bigram model with no zeroes is very expensive in the current representation. This is solved by augmenting the representation with a set of base values for each row, where values which are unrepresented on a row are now assumed to be the base value for that row. Figure 6.3 shows a smoothed version of the matrix in figure 6.2

$$\begin{pmatrix} 20.1 & 0.3 & 0.3 & 0.3 \\ 2.2 & 2.2 & 2.2 & 35.4 \\ 40 & 3 & 3 & 3 \\ 3.8 & 3.8 & 3.8 & 3.8 \end{pmatrix}$$

is stored as

row	**column**	**value**
1	1	20.1
2	4	35.4
3	1	40

and

row	**base**
1	0.3
2	2.2
3	3
4	3.8

Figure 6.3: The augmented MATLAB sparse matrix format

represented in the augmented format.

The augmented representation allows only certain types of smoothing to be performed: only those smoothing techniques which assign the same probability mass to all the unseen bigrams in a particular row of the bigram matrix can be used. The modelling subsystem uses a two-step Good-Turing algorithm: firstly, each row in the bigram matrix is smoothed using the Simple-Good-Turing (SGT) algorithm. This gives an estimate of all unseen bigrams in which the context was seen. However, any row with no bigram counts whatsoever must be allocated some probability mass from the whole model. The second step is to smooth the totals of the rows using SGT, scaling down rows with bigram counts and redistributing the stolen mass to the ones without.

After smoothing, the bigram model is ready to be generalised using SVD.

6.3 Generalisation

This section describes the generalisation subsystem shown in figure 6.1(b), whose purpose is to take the models produced by the modelling subsystem and build generalised models using SVD. Generalisation can be broken down into two sub-stages, decomposition and approximation.

The decomposition sub-stage is a series of steps which analyse the bigram model and split the bigram matrix into singular vectors and values. The approximation sub-stage is almost a mirror image of the decomposition, synthesising generalised bigram matrices with the singular vectors and values, essentially doing decomposition in reverse: except that information is discarded when building the generalised matrices.

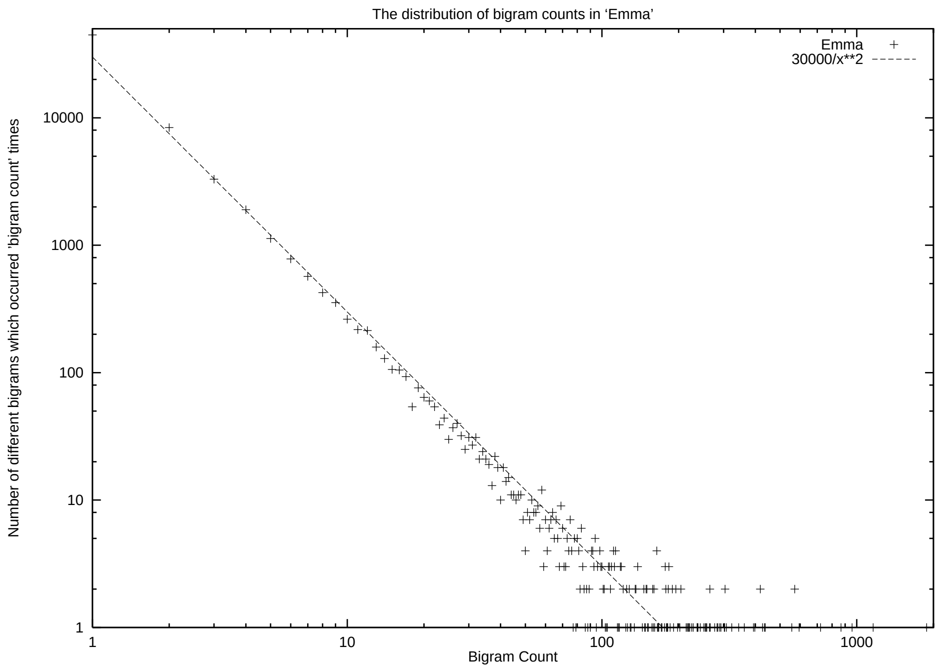

Figure 6.4: The distribution of bigram counts in "Emma"

6.3.1 Decomposition

The purpose of the decomposition sub-stage is to decompose the bigram matrix produced by the modelling subsystem into singular values and vectors. The decomposition sub-stage is made up of 3 steps:

1. "Squish" the bigram model to tone down extremely high values.

2. Make the bigram model sparse again in preparation for SVD.

3. Perform SVD.

Squishing

For Jane Austen's "Emma", the most frequent non-zero bigram count is one (there are 44,701 bigram counts of one), with higher bigram counts decreasing in frequency very rapidly (see figure 6.4). In fact, about 90% of all bigram counts are four or less, and only 0.24% of all bigram counts are greater than 100. This means that for the vast majority of bigram counts in a bigram matrix will be very small, but a few, such as the highest bigram count of 1,880, will be extremely large compared to most values.

This mixture of many low bigram counts and a few extremely high bigram counts is typical in word sequences, and could be detrimental to the approximation, since undue effort or freedom within the model may be used to model the few very high count bigrams and the smaller ones may get overwhelmed.

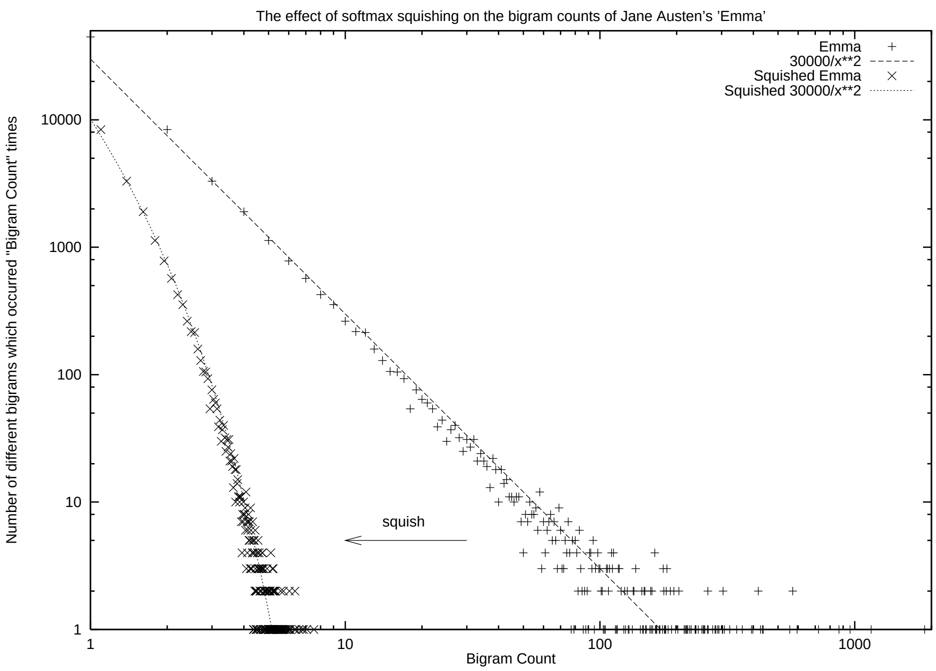

Figure 6.5: The effect of softmax squishing

To reduce the effect of the few very high counts, the models produced by the modelling subsystem are "squished" by a transformation much like the softmax method in chapter 3. In particular, for a bigram matrix X, the squishing transformation produces $\mathcal{X}$ such that:

$$\mathcal{X}_{ij} = \ln(X_{ij} + 1)$$

This particular function was chosen because it preserves order and ensures positive values remain positive. Figure 6.5 shows that the squishing is quite severe. For example: when the highest count in "Emma" of 1,880 is squished, it transforms into approximately 7.5, bringing the highest value down very close to the rest. Using the softmax squishing function means that SVD will not spend undue amounts of freedom modelling large counts, but it may be too severe. If this turns out to be the case, other functions, such as the square root, can be as easily used instead. Note that this step translates and transforms the matrix into log-space; which deforms the bigram matrix into something which no longer corresponds to a set of probability distributions. The squishing must therefore be reversed before probabilities are calculated with the model.

Translate to Zero

In the last step of the decomposition sub-stage we wish to use a sparse matrix SVD algorithm to decompose the bigram matrix. In the modelling subsystem, the counts in the bigram matrix were smoothed and augmented with base values to remove zero probabilities from the model. The base values of the augmented bigram matrix cannot be used by a sparse matrix SVD algorithm — sparse SVD algorithms leverage the high number of zeroes to improve efficiency, which is incompatible with the base values of the augmented matrix representation.

To produce a sparse bigram matrix, the generalisation subsystem translates each row down by the row's base value, this has the effect of making all unrepresented bigrams in the matrix zero once more. One side effect of this translation is that the bigram matrix is even further from a set of distributions, but is now a set of squished "count-differences" from the base values. Although this step seems to be an unfortunate step backwards in terms of keeping good distributions, the information still remains in the form of the count differences, which will be re-combined with the base values during the approximation sub-stage.

Note that the base values will not be approximated — only the "count-differences" in the sparse bigram matrix will be approximated using SVD. Because the base values are not approximated it is important that the base values are good estimations to complement the approximated "count-differences" when they are combined to produce the generalised bigram models.

Sparse SVD

Word based bigram models of text are very large, and standard SVD algorithms do not perform well on such large structures. However, the bigram models produced by this system are very sparse, and this can be used to our advantage by using a sparse SVD algorithm, where the memory and computational needs are based the number of non-zero values in the matrix.

The system uses the las2 algorithm from the SVDPACKC suite of sparse SVD algorithms (see [BDO+93]). The las2 algorithm restates the problem of SVD as a equivalent eigenvalue problem and uses the "single-vector Lanczos method" to find the eigenvalues, which correspond to the singular values, of the matrix. The single-vector Lanczos method computes eigenvalues by generating a series of matrices, which have the property that their extremal eigenvalues are progressively better estimates of the extremal eigenvalues of the matrix being analysed.

The series of matrices produced by the las2 algorithm is generated recursively, with each matrix depending directly on the previous matrix. The finite precision of computers limits the accuracy of this recursion and the matrices need constant reorthogonalisation with respect to all previous steps in the recursion. This reorthogonalisation is an expensive process and is only partially done in the las2 algorithm for efficiency reasons. Because only some reorthogonalisation is done, errors creep in when trying to produce large numbers of singular values from the large matrices produced in the system. We have found that the maximum number of singular values which can be computed in a reasonable time is about 1,400.

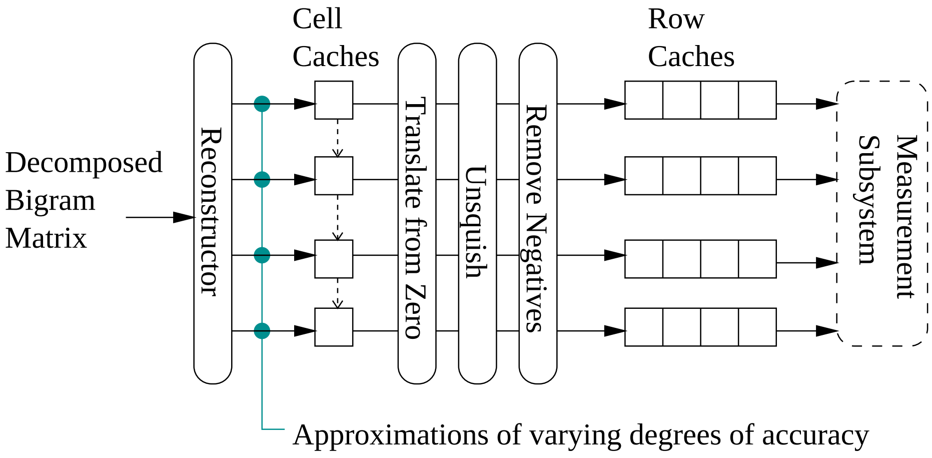

Figure 6.6: Reconstruction on demand caching scheme

6.3.2 Approximation

The approximation sub-stage produces a series of generalised bigram models based on the decomposed matrix produced by the decomposition sub-stage. The series of generalised bigram models is produced by reconstructing the decomposed bigram matrix using varying amounts of singular values, which is equivalent to approximating the bigram matrix to varying degrees of accuracy. By substituting the different approximated bigram matrices for the original bigram matrix produced by the modelling subsystem, a series of generalised bigram models is produced.

The system is capable of producing many generalised bigram models. If the SVD algorithm produces 1,400 singular values, it is possible to make 1,400 different bigram models; to save space, the matrix is only approximated to specified amounts of the singular values.

In principle, the series of bigram matrices are approximated in the fashion described in chapter 3. However the approximated matrices are unable to be stored in a sparse representation efficiently because the approximation will deform previously zero values independently of which row they are on. Therefore, a different scheme must be used to handle the large amount of data produced by the system. The solution used in the system is to only deal with one row of the reconstructed bigram matrices at a time by only reconstructing and storing a row when it is needed.

Figure 6.6 shows the caching and processing scheme used in the system to avoid storing all of the reconstructed bigrams at once. When a cell is needed by the measurement subsystem, cells for all the specified dimensions are computed and cached. The generation of all dimensions of a cell at once allows the system to leverage the property that the computation of a cell at one particular dimension is a subproblem of the computation of the cell at a higher dimension: this means efficient computation of the same cell for all the matrices in the series.

At a later stage in the system probabilities will need to be generated, requiring the total sum of all cells in a row. To aid in this computation the whole row of a matrix is generated and cached when a cell is requested. So long as all processing for a particular row is done at the same time, then the caching scheme will not cause any more computation than necessary and only one row of each matrix in the series needs to be stored at any one time.

Figure 6.6 also shows the steps of "translate from zero", "unsquish" and "remove negatives", their purpose is to reverse the preprocessing that allowed the sparse SVD algorithm to be used and to make the resulting matrix compatible with a bigram model.

Translate from zero

Translate from zero restores the adjustments made during decomposition to make the matrix sparse for the SVD algorithm. The translation is simple, for every row which is reconstructed, add the base value of the row to every cell in the row.

Unsquish

The unsquish step undoes the softmax squishing that was performed on the bigram counts to reduce the effect of very high bigram counts. For a reconstructed "squished" matrix $\mathcal{Y}$, the unsquish method transforms $\mathcal{Y}$ to Y where:

$$Y_{ij} = e^{\mathcal{Y}_{ij}} - 1$$

One problem with this step is that small errors in the approximation will be exaggerated by this process. In fact, for an error of size a during approximation on a bigram count b, the final error will be on the order of:

$$O(e^a b)$$

The exaggerated error is a result of the very severe squishing brought about by using logarithms, if this becomes a problem other squishing functions could be used to reduce this effect.

Remove Negatives

Chapter 3 described how an approximated matrix may contain negative numbers, and two solutions were proposed: exponential tail and softmax. For this step, the exponential tail method is used to remove negative values from the matrix: with the threshold set to the smallest bigram count in the row before approximation. Unfortunately, softmax method of removing negative values can not be used: it requires taking the logarithm of the bigram matrix before approximation, and can easily turn small fractions contained in the bigram matrix into large negative numbers that might interfere with the approximation.

6.4 Measurement

The purpose of the system is to build and test generalised bigram models. The system begins with a training set from which generalised bigram models are produced, and a test set to evaluate those models. The measurement subsystem uses the test set to evaluate the models by calculating the entropy of the test set using the predictions of each of the bigram models, as described in chapter 2. In principle, the evaluation of one generalised bigram model is very similar to the method used in the exploratory experiment described in chapter 5, but the size of the models requires special considerations during the design of the algorithm to compute the entropy of the test set.

6.4.1 Calculating the Cross Entropy

To generate the data needed to test the models, the test set is also put through the first three steps of the modelling subsystem. This builds a test matrix that contains counts of all the bigrams which occur in the test set. This is all the information needed from the test set to compute the cross entropy of the test set using the generalised bigram models.

The calculation of the cross entropy of the test set on a generalised bigram model is done in the same fashion as the exploratory experiment in chapter 5. The expression for the entropy of a bigram represented in the ith row and the jth column of a generalised bigram model is:

$$\text{entropy}(Y_{ij}) = -\log\left(\frac{Y_{ij}}{\sum_{k=1}^{c} Y_{ik}}\right) \tag{6.1}$$

Where Y is the bigram matrix of a generalised bigram model, and c is the number of rows of Y.

The expression for cross entropy of the test set on a generalised bigram model is:

$$\text{cross entropy} = \sum_{i=1}^{r}\left(\sum_{j=1}^{c}(\text{entropy}(Y_{ij}) \times T_{ij})\right) \tag{6.2}$$

Where T is the test matrix, r is the number of rows in T and Y. Note that T is very sparse and only values of T which are non-zero need to be computed. The computation of cross entropy is therefore more tractable than it first appears.

The measurement subsystem calculates cross entropy of the test set for each member of the series of generalised bigram models. Having a cross entropy for each of the generalised bigram models allows a comparison between the different degrees of approximation, and will show the how the best generalised bigram models would perform when encoding the test set.

6.4.2 Special Consideration for Large Models

The generalised bigram models produced by the generalisation subsystem are very large and difficult to manipulate because the bigram matrices they contain are no longer sparse. The bigram matrices are no longer sparse because the approximation done using SVD does not make any effort to retain zero values, and will be unlikely to approximate zero to be exactly zero.

Because the bigram matrices are so large, the measurement subsystem deals with only one row at a time from each of the bigram matrices contained in the series of generalised bigram models. Formula 6.1 shows that the entropy computation for one bigram requires the entire row of the bigram matrix; by reconstructing a whole row at a time, all the information needed to compute the entropy of any bigram represented in that row is made available.

The calculation of cross entropy is done in row order by sorting the test set by row. Figure 6.2 showed how an unsmoothed bigram matrix is represented as a list of non-zero bigram counts and their position in the bigram matrix, so sorting is needed if the bigrams are stored in an unordered fashion. The order is important because the caching scheme in the generalisation sub-stage is designed to reduce the storage requirements of the reconstruction of a generalised bigram model by only ever reconstructing and storing one row at a time. By sorting the test set by row, all the entropy calculations requiring a row are grouped together, ensuring that the caching scheme only needs to reconstruct any row once, and can then discard it when a new row is needed.

The algorithm for computing cross entropy

To test the generalised models, the measurement subsystem processes the test set in row order. For each bigram which occurs in the test set, the entropy of the bigram in the generalised model is calculated using formula 6.1 and multiplied by how many times it occurred in the test set.

When the entropy of a bigram is calculated, the modelling subsystem requests that the bigram count of the bigram in question be reconstructed from the generalisation subsystem. The caching scheme in the generalisation subsystem reconstructs and caches the entire row of bigram counts if they have not already been cached, essentially doing all the reconstruction needed to calculate the entropy of every bigram in the row.

The entropy of all the bigrams which occur in the test set are added together to form a total entropy measure of the test set using the bigram. Note that this algorithm is a tractable implementation of formula 6.2 that ignores all bigrams of zero count.

6.5 A Separate System Which Does No Approximation

A bigram matrix which is approximated using all the singular values which exist in the original matrix is a perfect reconstruction, and is identical to the original matrix. However, it is not tractable to compute all the singular values of the large bigram matrices used in the system. By using the original matrix in place of the perfect reconstruction, the total entropy of the bigram model generalised using the maximum number of singular values can be computed. Because the original matrix is sparse and is already represented in a sparse format, a separate module (not shown in figure 6.1) is used to calculate the total entropy on the smoothed bigram model produced by the modelling subsystem.

Chapter 7

Experiments

The previous chapter described a system to build and test generalised bigram models of written English. This chapter presents a series of experiments designed to test the performance of the system on word-bigram models of text.

There were three experiments:

Experiments with artificial hidden Markov models :
The system described in chapter 6 introduces new stages in the method of building bigram models and generalising them using SVD. To test the basic functionality of the new stages, the system was used to evaluate the performance generalised bigram models based on data generated by hidden Markov models (HMMs).

Experiments using real text :
The end goal in this thesis is to build better models of English text. The system described in chapter 6 is purposely designed to evaluate the effectiveness of generalising large bigram models with very sparse data, such as found in word based bigram models of written English. We have carried out a series of experiments using different texts to see if bigram generalisation using SVD can improve a bigram model of written English.

Experiments with "cleaned" text :
A common strategy when modelling the words of English text is to remove capitalisation and punctuation from the text to make it simpler to model. To explore the effect of such a scheme on the performance of generalised bigram models, we repeated the experiments above after cleaning the data to remove punctuation and capitalisation.

7.1 Experiments with Artificial Data

The system described in chapter 6 differs from the method used in the exploratory experiment described in chapter 5. The system smoothes bigram counts to compensate for the lack of data present in a sequence of words derived from text, and also transforms the bigram matrix to allow a sparse SVD algorithm to be used. The exploratory experiment did not require any smoothing to the bigram matrix, nor did it need to transform the bigram matrix to make the use of a sparse SVD algorithm possible. We have performed a series of tests using the system to test that the method still improves small models, even with the additional manipulations.

This section presents an experiment to test the effectiveness of the system on data generated from randomly generated hidden Markov models (HMMs) that have repeated distributions.

7.1.1 Hypothesis

We hypothesise that the changes made to the system to cope with the large sparse bigram models of text will still allow the system to improve bigram models of HMM data with repeated distributions through SVD generalisation.

7.1.2 Method

The experiment was made up of twelve trials of the following structure:

1. Generate a hidden Markov model.
2. Produce three sequences, of which the first is used to derive 10 training sets of different lengths, and the last two are used as test sets.
3. For each training set, use the system to create as many different generalised bigram models as possible.
4. Use the system to compute an encoding length for the test set on all the generalised bigram models.

Generating HMMs

The twelve 5-state hidden Markov models like the one in Figure 7.1 were generated using the following method:

1. Generate a transition distribution for each state by using a stable distribution. The same method was used to generate the distributions for the artificial bigram models from a stable distribution in chapter 5.
2. Pick two symbols which are not produced by any other state and assign them probabilities of 0.5 in the output distribution. This will mean that each state transition will generate four bigrams with the same distribution.

Transition probabilities:

current node	next node 0	1	2	3	4
0	0.083	0.250	0.083	0.083	0.500
1	0.125	0.125	0.375	0.250	0.125
2	0.125	0.125	0.125	0.250	0.375
3	0.125	0.250	0.250	0.125	0.250
4	0.111	0.444	0.222	0.111	0.111

Output probabilities:

	Output Symbols									
node	A	B	C	D	E	F	G	H	I	J
0	0.5	0.5								
1			0.5	0.5						
2					0.5	0.5				
3							0.5	0.5		
4									0.5	0.5

Figure 7.1: Markov model used in figure 7.2

Generating training and test sets

The hidden Markov models were used to generate three sequences of 5000 symbols each. Ten training sequences were derived from the first sequence, consisting of the first 8, 16, 32, 64, ..., 4096 symbols of the sequence respectively. The second and third sequences of 5000 were used as test sets.

Doing the Test

For each of the twelve Markov models, the system was run 20 times, testing the bigram generated by each of the data sets against each of the test sets. The average entropy of each run was then plotted in a graph like that shown in figure 7.2. Also plotted is the average entropy of the bigram before generalisation. Note that the final point plotted on each line is the entropy of the test set with respect to the ungeneralised bigram model.

7.1.3 Results, Analysis, and Conclusions

Figure 7.2 shows the performance of generalised bigrams on data generated by the HMM shown in figure 7.1. The results shown are typical of all of the different markov models tested.

For small sets of training data (between 8 and 32 symbols — the upper curves in figure 7.2), generalised models provided a significant improvement over the ungeneralised bigram model by generalising using only one dimension. A one dimensional generalisation is equivalent to dropping down to a unigram model.

For some small training sets, the SVD algorithm was unable to find 10 singular values. The lack of singular values is not an error: it is a symptom of the very sparse nature of the model when trained on very little data.

When the models were trained on middling amounts of training data (between 64 and 512 symbols), generalised bigram models provided a significant improvement at 3 dimensions. 3 dimensions is lower than the number of different distributions in the HMM used to generate the data, and so the generalisation may be doing a mixture of modelling individual distributions and overall token probabilities. Alternatively, there may be a few distributions which are similar enough to be modelled sufficiently well by only one distribution.

When the model was trained on data sets of 1024 symbols or higher, generalised bigram models provided only a small improvement over the ungeneralised model.

When the bigram model was trained on data sets of 2048 symbols and 4096 symbols, a minima occurred at 5 dimensions — the exact number of distributions in the original model. This leads us to conclude that the system can still detect and represent the number of distributions in the model.

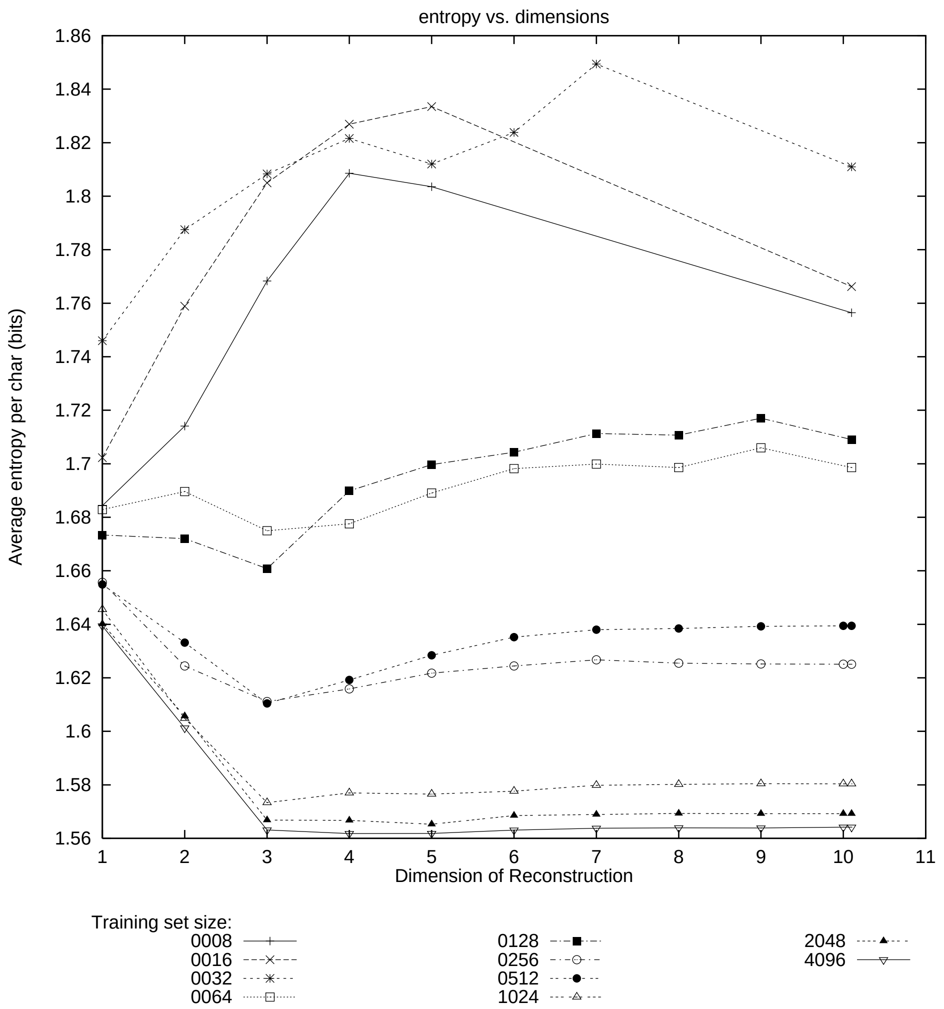

Figure 7.2: Testing different sizes of training data

7.1.4 Summary

The system can improve bigram models of small artificial bigram models such as the ones built in the experiment. The system can generalise low-count bigram models to lower order models (eg the unigram) when it is better to do so. Furthermore, the system can provide a smooth transition to the lower order model, a property which may be useful for a more complex type of model (eg a variable order Markov model). The system has retained the ability to detect the number of distributions used to generate the sequence when there is enough data. The systems' effectiveness is not as great when there is enough data to make a good bigram model.

7.2 Experiments Using Text

The experiment on HMM data showed that using SVD produced improved bigrams models for data generated from simple HMMs. It also showed that it was still possible to identify the exact number of distributions contained in the HMM. This section describes an experiment which uses the system to build and generalise a bigram model of the sequence of words contained in a piece of written English. The aim of the experiment described in this section is to see if bigram models of written English can also be improved through SVD generalisation.

This section details three experiments performed in three different pieces of text:

1. "Emma" by Jane Austen.

2. The Complete Works of Jane Austen.

3. The Bible (King James Version).

7.2.1 Hypothesis

We hypothesise that SVD generalisation will improve a bigram model of the sequence of words contained in written English.

The hypothesis is an extrapolation from the conclusions drawn from the previous experiment, where generalisation using SVD was able to improve a bigram model built on data generated from a five state HMM. Furthermore, the HMMs were deliberately constructed to have properties in common with the structure of English. The transition probabilities of the HMMs in the previous experiment were generated using stable distributions. This resulted in distributions with mostly small probabilities and the occasional very high one. We believe the transition distributions contained in the HMM have similar structure to those found in word-based bigram models of text, and bigram generalisation using SVD should therefore be successful in making better models.

7.2.2 Method

The experiments evaluated the performance of generalised bigram models on three different data sets. The evaluation was done in the following steps in all three cases:

1. Clean and divide the text into a test set and training set.

2. Use the system to evaluate the performance of bigram model generalisation with the training and test sets.

The three data sets were:

"Emma". The novel "Emma" by Jane Austen was obtained from the Project Gutenberg web-site and stripped of the standard Gutenberg headers and footers. No other cleaning

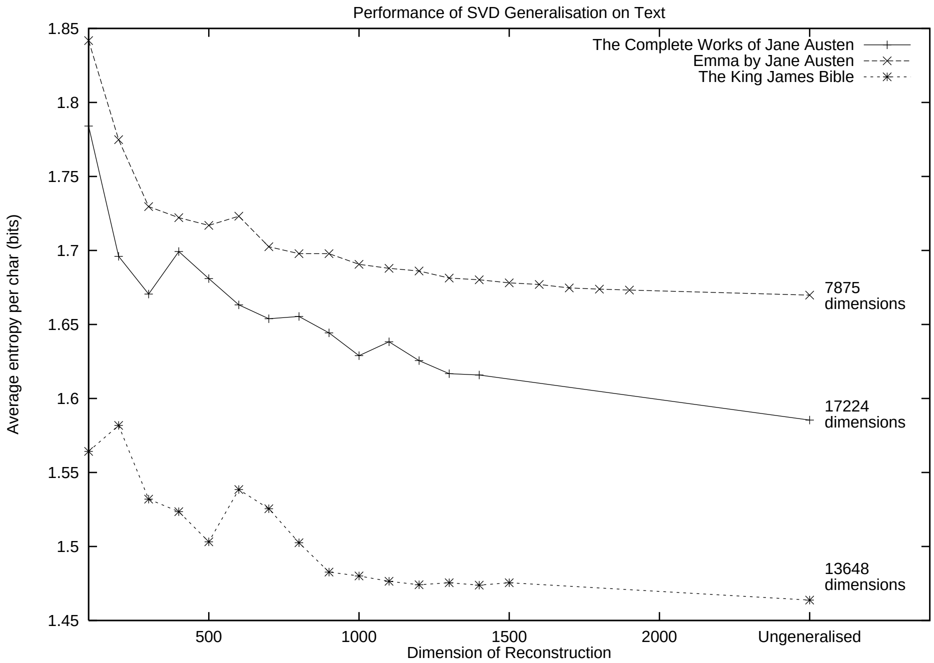

Figure 7.3: Encoding the last part of a body of text using the rest of the text

was done for "Emma". The training set is comprised of the first 27 chapters, and the test set is the last 2 chapters.

The Complete Works of Jane Austen (CWJA). The Complete Works of Jane Austen (CWJA) was obtained from Project Gutenberg and other web sites. It is a concatenation of all the novels written by Jane Austen and includes "Emma". All additional headers and footers were removed from the texts before concatenation. No other cleaning was done for CWJA. For CWJA the test set was the last 10371 characters of the text, and the rest was used as the training set.

The Bible. The King James Bible was again obtained from Project Gutenberg. The Gutenberg headers and footers were removed as usual but also the verse and book numbers which precede every verse were also removed. The test set of the Bible data set is the last 9596 characters of the Bible data set, with the rest used for training.

7.2.3 Results, Analysis and Conclusions

Figure 7.3 shows the results of all three tests.

In each of the three tests the system was not able to produce enough singular values to produce a model that even equalled the performance of an ungeneralised bigram model. The maximum number of singular values used in the experiment are at the limit of the SVD algorithm used in the system. When called upon to produce more singular values, the SVD algorithm was unable to converge in a reasonable amount of time.

The Bible was able to be modelled the best out of all of the three data sets, followed by the CWJA set, and then "Emma".

Although CWJA is the biggest and would therefore be a more likely candidate for the lowest entropy, the bible is a far simpler data set in terms of the number of words compared to its total length. In similar trials using other methods [Tea98] this was also the case.

The extra complexity shown in the entropy measure of CWJA is also supported by the rate at which the entropy is descending. For "Emma" and the Bible, a levelling off of the descent is clearly visible, but with CWJA, no such effect is easily seen.

To check the performance of the system more rigorously, the "Emma" data set was put through more tests. A ten-fold cross validation was done using the "Emma" data set, but the performance did not differ through each validation.

It is impossible to tell for certain whether the generalised models would out-perform the ungeneralised models if enough singular values were used. The rate of descent of each graph is quite steep at lower dimensions, with what appears to be a levelling section at the end of the Bible and "Emma" graphs.

Although the system was not able to produce a model that was an improvement over the ordinary bigram model, this does not mean that the method will not work with the ability to generate more singular values.

However, a close inspection of the bible results reveals a very flat improvement over the last 500 singular values. This suggests that no more significant gains can be made by including more singular values in the model and that it will just gradually converge on the performance of the ungeneralised bigram model.

One can argue that even if the system could generate more than 1,400 singular values, the performance would not exceed that of the ungeneralised model. With 1,400 dimensions, there are more parameters in the model than the total number of bigrams in the training set. For example: if there are 10,000 different words in the model, and 1,400 singular values are produced, then the U and V matrices of the SVD decomposition will each contain 14,000,000 values. This suggests that additional singular values would not improve performance. However, it may be that although there is a large amount of freedom in the model, most of the freedom is used to model zeroes — there are many cases where word transitions do not occur, and the model must represent these as well. If that is the case, then it is possible that SVD generalisation can improve upon the results of an ungeneralised bigram model.

7.2.4 Summary

This experiment is suggestive that SVD generalisation does not help to improve bigram models of word sequences contained in written English. But there remains some doubt because of the limitations of the SVD algorithm.

The fact still remains that the system was not able to produce a model which is better than even the ungeneralised bigram model, as such we can only conclude that it is unlikely that bigram generalisation using SVD will improve bigram models on this task.

7.3 Experiments with Cleaning

The previous experiment produced discouraging, but inconclusive results as to whether SVD generalisation can improve a bigram model of word sequences contained in written English. The inconclusive results were the direct result of the limitations of the SVD algorithm, which could not produce enough singular values to show if SVD generalisation would eventually provide a better model or not.

The goal of the experiment described in this section is to allow a greater proportion of the total number of possible singular values to be produced. By cleaning the training and test texts more vigorously, the sequence of words will become simpler, and the number of singular values required will become less.

7.3.1 More Cleaning

There are different kinds of cleaning able to be done to a piece of text to reduce it's complexity. In the experiment described in this section, we used two of the most common methods: removing capitalisation, and removing all punctuation.

Removing capitalisation simplifies the data, words that are capitalised because they either start a sentence, or are proper nouns, will no longer be differentiated from their lower case counterparts. Removing capitalisation therefore reduces the number of words and complexity of the text in one easy transformation which does not change the meaning of the text too much. However, the differentiation of proper nouns with the same spelling as other words becomes very difficult. One solution would be to keep all proper nouns capitalised, but this again is difficult without a sophisticated tagging system or a list of proper nouns found in the text.

Removing punctuation from the text will reduce the complexity of the text. Punctuation symbols are meta symbols of English text: they can delimit and connect logical constructs, and modify the meaning of sentences and words. A system to model a sequence of meanings could drop these markers in favour of a simpler sequence to model, but some information and meaning of the text will be lost.

The assumption made by removing punctuation and capitalisation from the text is that most meaning and structure of English is contained in the words. A system that can model the word sequence of English could then be extended to include punctuation at a later date. However, punctuation is a large part of written English and punctuation symbols are far more common then most words. Any method or system of prediction must therefore be wary of making such an assumption, especially when comparing performance with models which may not make such an assumption.

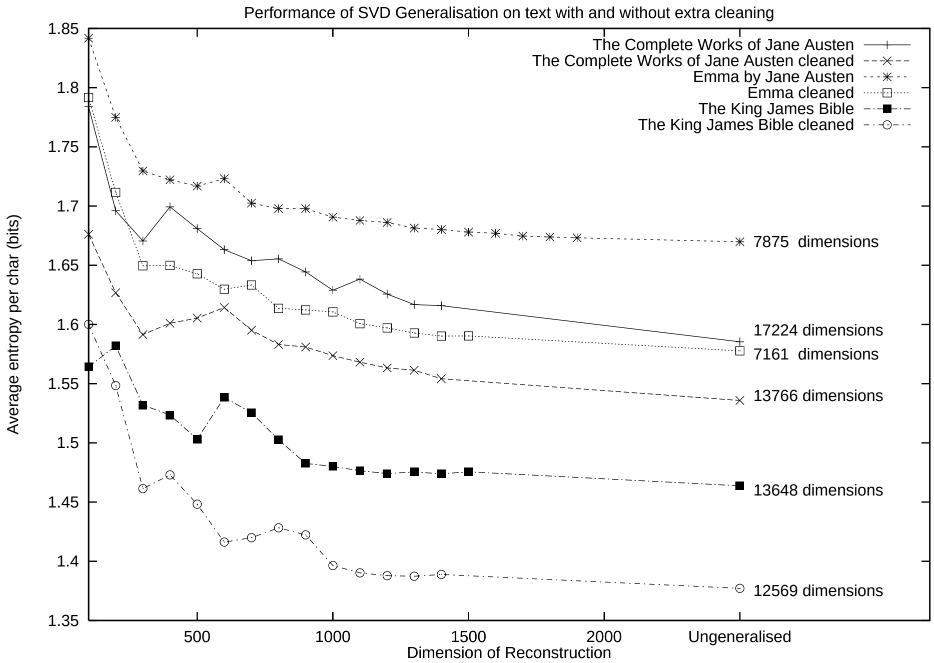

Figure 7.4: Tests using cleaned data

7.3.2 Hypothesis

We hypothesise that the reduction in complexity through cleaning will make it possible to build a generalised bigram model better than an ungeneralised one. However, we would note that the previous experiment done on the Bible text showed that although the Bible was the easiest to model, no improvement over the ungeneralised bigram model was able to be found. The previous experiment also showed little promise in using more singular values in a generalised bigram model.

7.3.3 Method

The experiment was done in the same fashion as the experiment described in section 7.2. The only difference is that the training and test sets were put through an additional stage of cleaning, where all capital letters were transformed to lower case, and all non-alphanumeric characters were removed and replaced with word boundaries.

7.3.4 Results, Analysis and Conclusions

The results of the experiment are shown in figure 7.4, along with the results of the experiment with less cleaning for comparison.

The performance of the bigram models produced using the additional smoothing consistently outperform the less-cleaned models. The uncleaned model's performance even follows the same pattern or shape of the less smoothed models, but at a lower entropy.

The shapes of the graphs show that the reduction in complexity does not allow the SVD generalisation to behave any differently, and to produce a model which out performs the ungeneralised bigram model. The reduction in entropy indicates that the true structure of more smoothed text is quite easier to model, and that the smoothing is a significant factor in the performance of these models.

7.4 Summary

Generalising bigram models of word sequences using SVD does not appear to be a useful thing to do for predicting large pieces of English text. The experiments show that although the method works for smaller and simpler artificial data, this does not extend to the larger and more complex problem of predicting the sequences of words contained in written English.

Chapter 8

Conclusions

We hypothesised that bigram models of written English words had inherent redundancy: because many words are used in the same way, many distributions in the model would be similar. We believed that by using SVD to generalise the bigram model, improvements could be made by removing this redundancy.

We proposed a new method of generalising bigram models using SVD to take advantage of this redundancy. This involved taking a bigram model, converting it into a matrix, extracting feature vectors using SVD, and generalising to varying degrees by using different numbers of dimensions of the feature vectors to build a generalised bigram model. By using SVD to generalise a bigram, the goal was to remove noise and combine information inside the model to get better models for predicting sequences.

We carried out a small exploratory experiment to test the method on a simple task of generalising artificial bigram models that consisted of repeated distributions. The experiment indicated that we were able to exploit the redundancy of bigram models.

We implemented a system to apply the method to bigram models that model the sequence of words contained in written English text. The system consisted of three subsystems:

- A modelling subsystem — to build a basic bigram model from the text as a base model for generalisation.
- A generalisation subsystem — to produce generalised bigram models.
- A measurement subsystem — to measure the effectiveness of the bigram generalisation.

The system was purposely designed to handle the large bigram models produced from English words. Measures such as cleaning the text, smoothing to provide better frequency estimates, and sparse representation of a bigram model are all implemented in the system to cope with a sequence English words.

On artificial data, the method worked well. The simple bigram models in the exploratory experiment were able to be generalised using the actual number of distributions they contained. On the system built specifically for bigram models of English words, data from simple

HMMs were able to be generalised to the point at which optimal performance was achieved by using the bigram models generated to the same number of dimensions as the number of states in the HMM.

On real data, the method does not appear to work. The results from applying the method to real text showed no improvement over a bigram model without any generalisation. Note however that the results are inconclusive — the number of dimensions produced by the SVD generalisation fell far short of the total number possible. Without all of the dimensions, not all of the possible levels of generalisation were able to be produced.

Although inconclusive, the results of using the method on real text also suggested that a greater number of dimensions would not help. For the Bible and "Emma" datasets, the gains from using higher numbers of dimensions were small, suggesting no more significant gains would appear.

It would seem that the method was not able to improve bigram models of English word sequences because of the greater complexity of real text. Even though artificial data suggested that the method would work, the size and complexity of real text is far greater than that of the artificial data created.

One possible reason involves the number of words. Bigram models of English word sequences can easily contain several thousand words, whereas the artificial models used to test the method only had about 10 different symbols at most. Because of the increased scale, SVD may not be able to capture the right components of the distributions contained in the bigram model to successfully generalise the model.

Another possible reason involves the nature of the redundancies. It should be noted that the redundancy of the bigram model of English word sequences is different to that of the artificial models used. The artificial models had distributions that were repeated *exactly*, this would be uncommon for a model of English word sequences, which are more likely to have only parts of their distributions repeated with parts of other distributions. This property is within the capabilities of SVD to generalise, but it is likely to be harder.

8.1 Future Work

There still remain some avenues for exploration for bigram generalisation using SVD. The main difficulty in the experiments for this thesis were the size of the bigram models used. The algorithm for performing SVD was pushed to its limits in the text experiments, and could not produce enough dimensions. By splitting apart the models and generalising the parts individually, the SVD algorithm will not need to be used to its maximum capacity. There are two obvious ways to split up the bigram model: by statistical reliability, and by the word type of the context.

Bigram models are a collection of distributions, each of which represents how likely other words appear after a particular word. By dividing up the distributions of a bigram model by how much information is available, we will be able to treat distributions with low statistical reliablility differently to those with higher reliability. Evidence that this scheme will work is provided by the results in the experiments with HMM data in section 7.1. The HMM experiment showed that bigram models with low statistical reliability needed to be generalised more than those of high reliability.

Another approach to dividing up the bigram model is to divide the model by word-type of the context symbol. The English language has much structure at the word-type level, such as adverbs are more commonly followed by verbs rather than another adverb. By leveraging this extra knowledge we can group the distributions of the bigram model and generalise each group separately. It is the assumption of this scheme that word types are more likely to share common usage and the distribution of words which follow should therefore be more similar than those of other word-types.

The cost of generalising the bigram models in parts is that correlations can no longer be made between the parts. By splitting up the model by statistical reliability, it is no longer possible to compensate a distribution with low reliability with that of a higher one. Splitting the model by word-parts may not suffer as much, but there would still be some loss as words can have multiple uses which may cross the boundaries of the split model.

Bibliography

[BDO$^+$93] Michael Berry, Theresa Do, Gavin O'brien, Vijay Krishna, and Sowmini Varadhan. SVDPACKC (version 1.0) user's guide. Technical Report CS-93-194, Department of Computer Science, University of Tennessee, April 1993.

[Che96] Stanley F. Chen. *Building Probabilistic Models for Natural Language*. PhD thesis, Harvard University, Cambridge, Massachusetts, May 1996.

[DDH90] Scott Deerwester, Susan T. Dumais, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society For Information Science*, 1990.

[Fer] John D. Ferguson. Hidden markov analysis: An introduction. An old typed paper of unknown origins.

[Goo53] I. J. Good. The population freqencies of species and the estimation of population parameters. *Biometrika*, 40:237–264, December 1953.

[Kat87] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(3):400–401, March 1987.

[LD97] Thomas K. Landauer and Susan T. Dumais. A solution to plato's problem: The latent sematic analysis theory of aquisition, induction and representation of knowledge. *Psycological Review*, 104(2):211–240, 1997.

[LLF] Thomas K. Landauer, Darrell Lahman, and Peter Foltz. Learning human-like knowledge by singular value decomposition: A progress report.

[Mat99] The MathWorks, Inc., 24 Prime Park Way Natick, MA 01760-1500. *Using MATLAB*, January 1999.

[PFTV89] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipies, The Art of Scientific Computing (FORTRAN Version)*. Cambridge University Press, 1989.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence a modern approach.* Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, New Jersey, 1995.

[Tea98] W. J. Teahan. *Modelling English Text.* PhD thesis, Univeristy of Waikato, 1998.