A Generative, Adaptive Language Model Approach to Spam Filtering

Ben Medlock, Masters Thesis, July 2003

This thesis is submitted in part fulfilment of the University of Cambridge MPhil degree in Computer Speech, Text and Internet Technology.

Word count: 14,990

Abstract

In this thesis we present a generative classification model for structured documents based on statistical language modelling theory, as well as introducing two variants of a new discounting technique for higher-order N-gram LM's. We apply the model to the spam filtering domain using a new email corpus assembled for this work and report promising results. We also present the best results achieved to date on the *LingSpam* email corpus (Androutsopoulos et al, 2000a).

Table of Contents

Table of Figures

Ch 1. Introduction

In this chapter, we introduce the motivation behind the work described in this thesis.

1.1 The Problem of Spam

According to the internet filtering company *MessageLabs*[1], the amount of unsolicited email sent via the internet in May 2003 exceeded the amount of genuine email by around 10%. Estimates reported by *Ferris Research*[2] suggest that such email will cost US companies around $10 billion in 2003, with the figure for UK companies standing at around £3 billion. Statistics such as these emphasise the enormity of the problem of unsolicited email (also known as *junk-mail* or *spam*), a problem which has grown steadily with the increasing widespread use of email, and shows no sign of diminishing.

As well as costing businesses billions in terms of wasted resources (network bandwidth, disk space etc.) and personnel time, unsolicited email carries the further risk of exposing young email users to offensive and psychologically harmful material.

Sending large volumes of spam is an extremely cost-effective method of advertising. For instance, if a 'spammer' sends 10 million messages and receives only a 0.001% response rate, that represents 1000 responses. No other form of advertising offers such widespread exposure for such a negligible initial outlay.

Laws governing the dissemination of unsolicited email vary from nation to nation and are notoriously difficult to enforce. Many governments are making new efforts to legislate against 'spammers', but it is unclear how successful these measures will be, as they are constrained by the need to uphold private email users' rights. Additionally, spam disseminated from countries that do not have anti-spam laws in place is particularly difficult to curb.[3]

[1] See http://www.messagelabs.co.uk/home/default.asp
[2] See http://www.ferris.com
[3] See http://news.bbc.co.uk/1/hi/technology/3005757.stm also http://www.cauce.org and http://spam.abuse.net

Spam comes in many forms, though the most prevalent are product advertisements and pornography. The following chart, published by *Brightmail*[4], shows a breakdown of common categories:

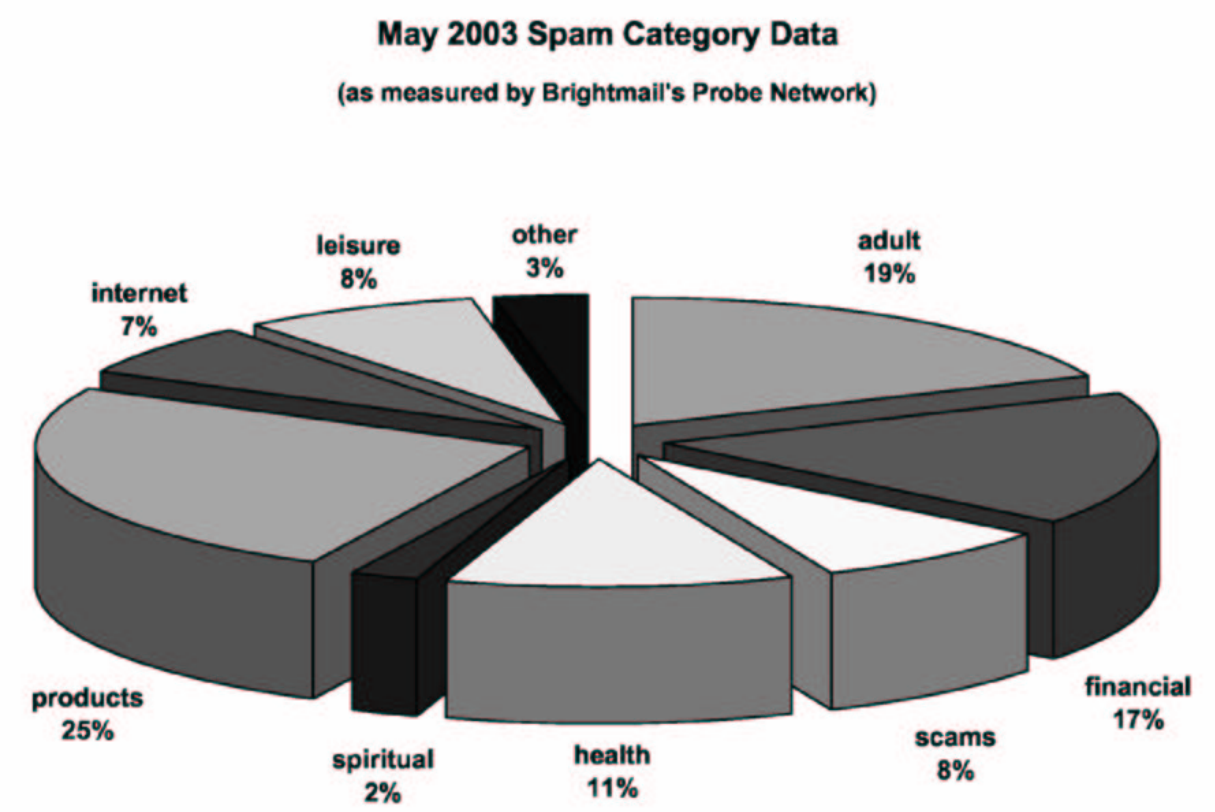

Figure 1.1: Spam category breakdown

An ideal solution to the problem of spam would be to curtail the ability of the spammer to send large quantities of unsolicited mail for virtually no cost. However, such a measure would involve significant changes to current email infrastructure, and would be likely to impinge on perceived private email users' rights. In the long term, this may be the optimal solution, but for now alternative methods of dealing with spam must be found.

If the dissemination of spam cannot easily be stopped, the next best approach is to filter it out upon receipt. This doesn't solve the problem of wasted physical resources, but can save individuals and organisations large amounts of time otherwise spent manually weeding out spam from genuine email. If effective, such filtering approaches also protect young email users from offensive material. In principle, widespread effective filtering minimises exposure to spam, and therefore goes some way toward negating the appeal of spam as a means of mass-advertising.

[4] See http://www.brightmail.com/spamstats.html

1.2 Approaches to Spam Filtering

Originally most spam filters were based on *blacklist/whitelist* approaches, in which email is either unconditionally blocked or accepted depending on whether the sender is on the blacklist or whitelist respectively. The blacklist approach was quickly found to be inadequate due to the ease with which spammers are able to send messages from accounts set up and used only once, rendering them undetectable to a blacklist. Whitelist approaches were also found to be inadequate due to their restrictiveness and general inflexibility.

Another common approach to spam filtering is to set up lists of keywords and rules, perhaps incorporating *regular expressions*, that are used to identify and filter spam. Such approaches can be quite successful if significant effort is put into generating a wide-coverage keyword and rule set, but they have a couple of major disadvantages. Firstly, they are highly susceptible to being 'tricked' by a clever spammer devising methods of evading the rules, and secondly they require regular maintenance as new rules and keywords must be incorporated to keep up with the changing face of spam.

More recently, statistical methods have been used to analyse and filter email. This turns the spam filtering problem into a *classification problem*, on which various well-developed classification techniques can be used, based on statistical information learned from actual email data. More specifically, the spam filtering problem is usually seen as a *text classification problem*, on the basis that most email contains some form of textual content. It has also been proposed that machine learning techniques be used to rapidly adapt the statistical parameters over time to compensate for the ever-changing nature of spam. These techniques appear to be quite successful, though the lack of sizeable, standardised test data has limited their rigorous evaluation and comparison (see next section).

In chapter 2 we look at various methods proposed in the literature for spam filtering (and more broadly text classification) based on statistical and machine learning techniques.

1.3 Data

As mentioned earlier, the lack of widely-available training and test data for the spam filtering problem has limited the thorough evaluation and comparison of proposed techniques. In light of this, it has been one of the aims of this project to begin developing a corpus of standardised spam and genuine email data on which different techniques can be rigorously evaluated. One of the major issues to be confronted in the assembly of such an email corpus is the preservation of donor privacy, due to the personal nature of genuine email. In chapter 3 we present and discuss the *anonymisation* procedure used to protect genuine email donors, and the issues faced in the assembly of the test corpus.

Ch 2. Literature Survey

In this section we briefly review the literature on text classification, as well as various applications of classification techniques proposed for statistical spam filtering.

2.1 Text Classification

The field of text classification (TC, also known as text *categorisation*) has received a great deal of interest over recent years. The rapid growth in online text resources has prompted research into techniques for the automatic organisation and retrieval of online documents, providing many applications for TC. Perhaps the most common has been that of categorising documents by *topic* (e.g. Lewis and Ringuette 1994, Wiener et al. 1996), though many other applications have also been considered, such as classification by *genre* (Kessler et al. 1997) or *sentiment* (Pang et al. 2002).

2.1.1 Generative vs. Discriminative

The focus of almost all recent text classification research has been on statistical and machine learning methods. Such approaches are usually classed as either *generative* or *discriminative*, depending on the form of the underlying model on which they are based. The generative approach models class-conditional probabilities from training data, and then applies *Bayes decision rule*[5] (or similar) to select the most probable. The discriminative approach, on the other hand, learns a classifier directly from the training data by identifying a discriminating function in the feature space. The popular view, borne out by arguments such as those articulated by Vapnik (1998), is that discriminative techniques are more effective, as they attempt to solve the classification problem directly. As a result, the majority of research has gone into developing discriminative approaches, with less attention given to their generative counterparts. However, results published by Rubinstein and Hastie (1997)[6] and recently by Ng and Jordan (2002) suggest that the situation is not as clear as is often believed, and that in some conditions generative or combined models can outperform discriminative ones. According to Ng and Jordan (2002), the relative success of the two approaches often depends largely on the amount of training data available.

The discriminative approach can be divided into two phases: *feature selection* and *classification*. The generative approach has analogues to these phases, which we discuss in detail when outlining our classification model (section 4).

2.1.2 Feature Selection

In text classification terms, a feature is some attribute chosen to represent a source item of text. The motivation behind the various feature selection techniques is to extract features that are good representatives of the particular information contained in the source from which they are drawn. Many text classification techniques consider only *single terms* as features, with a *native feature space* consisting of all unique words that occur in documents of the type being classified (such techniques are often known as *bag of words* approaches). The feature space may contain tens of thousands of terms, and for many

[5] See section 4 for further information.
[6] Rubinstein and Hastie use the term *informative* instead of *generative*.

classification techniques it is highly beneficial to reduce the number of features, for the sake of computational tractability and discriminatory effectiveness. Yang and Pedersen (1997) compare five methods of selecting single terms as features, based on various metrics for judging the 'goodness' of potential terms. They focus on aggressively reducing the feature space and their results suggest that *information gain*, *chi-squared testing* and *document frequency thresholding* are good feature selection metrics. They also conclude that for text classification in general, common terms make better features than rare ones, contrary to the established view derived from research in *information retrieval*.

Of course, features need not be limited to single terms. It is well known that inter-term relationships are in principle more informative than single terms, though incorporating them robustly into classification models has often met with limited success. In the early 90's Lewis experimented with phrasal features for text classification (based on PoS tagging and noun-phrase extraction) and concluded that such phrases were in general *detrimental* to classification performance due to unreliable frequency distributions and increased noise (e.g. Lewis 1992). More recent work has been carried out suggesting that phrasal features can improve classification performance, although the results have been somewhat mixed, with Tan et al. 2002 reporting increases in true positives but also in false negatives, and Fürnkranz 1998 reporting some improvement but only for relatively short (2-3 word) phrases.

2.1.3 Classification

For the discriminative approach, the classification phase consists of learning a (usually linear) discrimination function in the feature space, whereas for the generative approach it consists of calculating class-conditional probabilities and applying Bayes rule to determine the most probable. Yang and Liu (1998) provide a thorough overview and statistical comparison of five popular classification techniques:

- Naïve Bayes (NB)
- Linear Least Squares Fit (LLSF)
- k-Nearest Neighbour (kNN)
- Neural Network (NNet)
- Support Vector Machine (SVM)

They conclude that all the techniques perform similarly given enough training data, but that LLSF, kNN and SVM outperform the others when limited training data is available. Of the five, NB is the only generative technique, and is shown to slightly underperform the others in general.

NB is a popular technique in much of the classification literature, due in large part to the ease with which it can be implemented and its computational simplicity. NB is based on the assumption that given a class, the conditional probabilities of all attributes (features) are independent of one another. The attribute probabilities are then simply multiplied together to yield the class-conditional probability. There are actually two commonly used NB models. The *multi-variate Bernoulli* model represents documents as vectors of binary attributes denoting term presence/absence, whereas the *multinomial* model takes term frequency into account, representing documents as sets of weighted terms. McCallam and Nigam (1998) discuss these two models in detail. It is interesting to note that in the spam filtering literature, an implementation of NB actually outperforms an implementation of kNN in a direct comparison (Androutsopoulos et al. 2000a – see below).

kNN is a classification technique widely studied in the pattern recognition community, performing well on a number of different classification tasks, including TC. The approach is quite simple: the document

to be classified, *D*, is mapped to the feature space, and the categories of its *k* nearest neighbours are weighted by the distance of each neighbour from *D*, aggregated and ranked as the most likely categories for *D*. Thresholding can then yield one or more categories to assign to *D*. Results using kNN for topic classification on the Reuters corpus have been presented by Han et al. (1999) and others.

LLSF is a discriminative classification technique introduced by Yang and Chute (1994), based on regression model learning. It performs comparably to kNN in experimental tests, though it is based on a different statistical model.

SVM's were introduced by Vapnik in 1995 and applied to the text classification problem by Joachims (1998). The idea behind the SVM approach is to learn the function which maximises the separation 'margin' between classes in the feature space. Yang and Liu (1998) point out that the SVM approach differs from many classification techniques in that only the training instances closest to the separation plane are used to determine the decision surface. This means that if all other training instances were removed the algorithm would still return the same results. Whether this has any actual bearing on classification performance has not been investigated. Another advantage offered by SVM's is the ability to use 'kernel functions' to map non-linear classifiers into a linear classification space.

Neural networks (NNet) have been extensively studied in the pattern classification literature, and applied to the problem of text classification by many researchers, including Weiner et al. (1995) and Ng et al. (1997). The tests carried out by Yang and Liu (1998) suggest that neural nets do not perform quite as well as the other discriminative classification techniques on TC and suffer from the disadvantage of being expensive to train.

2.2 Statistical Spam Filtering

Statistical spam filtering has been taken as an instance of a TC problem, although many have pointed out that it differs from traditional TC in that email contains many non-textual features that can in theory be used to improve categorisation performance. Sahami et al. (1998) note the presence of such features as over-emphasised punctuation (!!!!) and domain names (.com, .edu) which often provide clues as to whether or not the email is spam. Additionally, email has a richer composition than flat text, containing meta-level structure such as the fields found in MIME compliant messages (*Subject*, *To*, *From, Content-Type* etc.). The spam filtering domain also differs from standard text classification in that the categories (spam and genuine) are highly user-dependent, and thus any deployable strategy must be adaptive in order to satisfy specific end-user preferences.

Textual and meta-level data can be incorporated into a statistical framework, such as Naïve Bayes, and some of the first published work on such a technique was carried out by Sahami et al. (1998). They use a multi-variate Bernoulli NB model and incorporate 20 hand-crafted domain-specific attributes along with single term and phrasal attributes. With the inclusion of domain-specific attributes, the reported results are quite high (around 0.98 precision and recall); however the training and test sets are small (less than 2000 total messages), and not publicly available, thus making the experiments non-repeatable. Also, without the inclusion of the hand-crafted domain specific attributes, precision and recall drop to around 0.90 - 0.95 with over 5% of legitimate email classified as spam.

Androutsopoulos et al. (2000a) present the first results for spam filtering on a publicly-available corpus – the *LingSpam* corpus, consisting of 2412 genuine and 481 spam email messages. The genuine messages are drawn from a linguistics newsgroup, and are thus somewhat homogeneous in their content, though the authors claim a degree of content variation as a result of people using the newsgroup for more general communication as well as for linguistic discussion. Even so, the corpus is not really representative of the typical email user's inbox, and it can be argued that the homogeneity of the genuine email makes the classification task artificially simple. Additionally, the corpus is arguably too small for a wide range of statistical methods to be thoroughly evaluated. Despite its weaknesses, the *LingSpam* corpus has allowed a number of spam filtering techniques to be comparatively tested. Androutsopoulos et al. compare a multivariate NB classifier with a kNN variant classifier and present results of around 0.90 precision and recall. They also propose a weighted accuracy measure, *WAcc*, designed to account for the fact that misclassifying genuine email is more costly than misclassifying spam. We discuss the usefulness of this measure in some detail in section 5.1. Their implementation of NB outperformed kNN in the reported experiments.

Drucker et al. (1999) publish results comparing the use of SVM's with various other discriminative classification techniques on the spam filtering problem, with binary-featured SVM's and *boosting decision trees* performing best overall. They also note that performance was better *without* the use of a stop list, and that all features should be used rather than just a subset, due to the complexity of finding the optimal subset. Unfortunately the test sets they used are not publicly available so the experiments cannot be repeated.

Carreras and Marquez (2001) build on previous work, publishing improved results on the *Ling-Spam* corpus using boosting decision trees with the *AdaBoost* algorithm of around 0.98 F1 for spam. They also present similar results on another publicly available corpus – the *PU1* corpus (Androutsopoulos et al. 2000b).

Recently, a number of researchers have presented papers on spam filtering systems, but comparison is difficult due to the lack of an adequate benchmark corpus. Jason Rennie (2000) presents a spam filtering system, *ifile*, based on an NB model. In his article *A Plan for Spam* (2002), Paul Graham sets out his method of filtering spam using a feature set based on textual and meta-level data, and William Yerazunis (2003) presents his filtering system, CRM114, based on *sparse polynomial hashing* with reportedly very high accuracy results, though once again comparison and verification are conspicuously absent.

Ch 3. Assembling a New Email Corpus

We now discuss the approach taken to assemble the new email corpus, which we will dub *GenSpam*, considering in detail the *anonymisation* procedure.

3.1 Overview

The need for a sizeable corpus of heterogeneous genuine and junk email is evident from the lack of comparability in much of the literature on spam filtering. The corpus assembled for this project will eventually be made publicly available as a benchmark for comparing spam filtering techniques. The corpus currently consists of:

- 8018 genuine email messages drawn from around 15 sources. (20.42%)
- 31235 spam email messages extracted from *spamarchive.org*[7]. (79.58%)

The imbalance in the number of messages is partly due to the fact that it is more difficult to obtain genuine email. In time we hope the numbers of messages will be more comparable. In fact, the corpus is not actually as unbalanced as it may appear; on the whole, spam messages tend to be much shorter than genuine ones, so in terms of actual content, the ratio is more like (35% genuine / 65% spam). In addition, the spam data contains a significantly lower percentage of genuine textual content (spam often contains random sequences of characters designed to throw filters off). Thus, the ratio of real textual data is, at an estimate around (40% genuine / 60% spam).

It is easy to obtain large quantities of spam; we extracted our corpus from sections 10-29 of the *spamarchive.org* collection of donated junk-mail; however the disadvantage of using such a body of spam is that some of the meta-level information is lost (such as the true sender and recipient – due to the fact that the spam is forwarded by the original recipient to the archive). Ideally, we would like to find a sizeable collection of spam that has all the meta-level information preserved.

The obstacles to a widely-available corpus of genuine email data are due to the personal nature of such email. Clearly the identities of email senders and recipients must be protected, as well as those of persons referenced within email body text. To overcome this problem, we have developed an anonymisation procedure which is designed to remove references to entities within email text. This includes email addresses, web sites, postal addresses, person names, organisation names, etc..

3.2 Formatting and Anonymisation

We now discuss the various phases involved in preparing and anonymising the email data.

3.2.1 Data Extraction and Formatting

[7] See http://www.spamarchive.org

Before anonymisation, the email data is manipulated from its raw state into a usable format, preserving various meta-level features. Relevant information is extracted from the raw email data (usually in MIME format) and marked up in XML. Extracted fields include:

- DATE
- FROM
- TO
- SUBJECT
- CONTENT-TYPE
- BODY

All other information is discarded. It is worth noting that a deployed spam filter should consist of two phases – the first to extract text from the message (i.e. from the body, in whichever format, and from any attachments) and the second to analyse it. Because we are only interested in the analysis phase, we retain only plain text (extracted from HTML where possible), and discard all other forms of attachment (such as image, postscript, pdf etc.). However, we do retain meta-level information about the type of attachment, as this could be useful in building statistical models.

3.2.2 Identity Field Anonymisation

The FROM and TO fields contain the email addresses of the sender and recipient(s) respectively. We use finite-state analysis to retain only top level domain (TLD)[8] information. For US-based sites, the TLD is defined as the string of characters trailing the final dot, i.e. '`com`' in '`ben@yahoo.com`'. For non-US sites, it is defined as the final 2-digit country code, along with the preceding domain type specification, i.e. '`ac.uk`' in '`ben@cam.ac.uk`' or '`co.uk`' in '`freecomputers@flnet.co.uk`'. This allows for potentially useful analysis of high-level sending and receiving domains, without any individual identity traceability.

3.2.3 Part-of-Speech Tagging

The text contained in the SUBJECT and BODY fields is then tokenised and PoS-tagged using modified forms of the RASP (*Robust Accurate Statistical Parsing*)[9] tokeniser and tagger. The RASP PoS tagger is a state-of-the-art statistical HMM-based tagger. Using the CLAWS2 tag set[10] we identify potentially sensitive terms, such as proper names and numbers (telephone, etc.), based on an augmented lexicon and previously-trained tag transition matrix. The weakness of the tagger is that it is trained on the BNC (*British National Corpus*)[11], and the disparity between that and general email text results in higher error rates than would be expected if the tagger was trained on more domain-relevant data. Through experimentation, we estimate that the tagger recognises entity names with a recall of 90–95% and a precision of around 70%.

[8] See http://www.icann.org/tlds for further TLD information
[9] RASP is developed and maintained by the Universities of Sussex and Cambridge. See http://www.cogs.susx.ac.uk/lab/nlp/rasp/
[10] See http://www.comp.lancs.ac.uk/computing/research/ucrel/claws2tags.html
[11] See http://www.hcu.ox.ac.uk/BNC

3.2.4 Automatic Pattern Learning

Recent work on *named-entity recognition* and *information extraction* suggests that high-precision proper name identification relies heavily on analysis of context (e.g. Mikheev 1999 and 2002). Thus, in addition to the PoS tagger, we use finite-state, semi-supervised ML and boot-strapping techniques to learn commonly-occurring patterns of entity names, seeded by the output from the tagger.

The first step in the pattern learning phase is to identify names and numbers recognised by the tagger and standardise them. We also standardise individual characters, often used to denote first names, i.e. '`B_ZZ1 Medlock_NP1`' becomes '`&CHAR_ZZ1 &NAME_NP1`'.

For each entity name identified by the tagger, its left and right contexts are stored and similar contexts grouped and counted. For example, consider the following (tagged) extract:

```
^ Hi_UH &NAME_NP1 ,_, ^

^ Where_RRQ does_VDZ &NAME_NP1 live_VV0 ?_? ^
```

Note that '^' denotes a sentence boundary. Pattern analysis would result in the following patterns being learned:

```
1) LC<^ Hi_UH> RC<,_, ^>

2) LC<where_RRQ does_VDZ> RC<live_VV0 ?_?>
```

LC and RC denote left and right context respectively. The data is scanned for all occurrences of such patterns, and those that occur above a given threshold number of times are retained. The corpus is then examined for occurrences of learned contexts in which the word type is not restricted to entity names. For example, the following might occur:

```
^ Hi_UH people_NN ,_, ^
```

This would be recognised by pattern 1 above (assuming it had been retained). We then calculate a reliability score based on the ratio of proper name to any-word-type context instances, ranging from 0 to 1, given by the following formula:

$$reliability(LC, RC) = \frac{freq(LC < NP > RC)}{freq(LC < * > RC)} \quad (1)$$

Higher values (those approaching 1) are suggestive of contexts in which named entities are likely to occur, and lower-valued patterns (below about 0.5) can usually be discarded. We define a class of words that are 'replaceable' (i.e. where the tagger is more likely to have made a mistake), consisting mainly of the open-class PoS categories (see Appendix A)

We then consider all replaceable words within given contexts to examine the predictive power of higher-valued patterns. For instance, if a given pattern returns many words that are genuine common nouns, as opposed to proper names, then it is less likely to be useful for anonymisation, even though many entity

names may occur in that particular context. In this way, we learn patterns that are good predictors of proper name contexts.

We experimented with different context lengths, and found that taking two words on either side (sometimes weakened to a single word) was a reasonably good compromise between making the contexts too specific (longer contexts) and too general (smaller contexts).

These automatically-learned patterns also illuminate certain structure within email. For example, in our corpus, a commonly occurring pattern was:

```
^ &NAME_NP1 ,_, ^    =    LC<^> RC<,_, ^>
```

This highlights the fact that many emails begin with the name of the recipient followed by a comma, e.g.

```
Ben,
```

However, in many other cases a 'greeting term' will be used to begin the email, e.g.

```
Hi,  or  Greetings,
```

Such contexts are in fact highly predictive of proper names if such 'greetings terms' are excluded. By examining greeting and closing contexts learned from our corpus, we extract a list of common greeting and closing terms, which we then manually extend using regular expressions to capture further variants. This enables us to specify some of the automatically learned patterns as 'greeting and closing patterns' where non-proper name terms are only anonymised if they are both *replaceable* (Appendix A) and do not occur in the list of common greeting/closing variants.

Because all of the pattern analysis tools we developed are written in perl, the more reliable patterns can easily be extended using regular expressions. For example, pattern 1 from above can be extended to capture common variants:

```
1) LC<^ \w*_UH> RC<(,|!)_. ^>
```

This will recognise greetings, such as:

```
Hi Ben,
Hello Ben,
Dear Ben,
Hey Ben!
...
```

3.2.5 Pattern Application and Term Propagation

When a list of reliable patterns has been learned, they are applied to the data. The aim is to maximise potential for learning entity names that the tagger might have missed. To this end, there are two states of the data in which the learned patterns can be applied:

1) Without any anonymisation.
2) With tagger-identified proper names, numbers and characters anonymised.

Recall that the pattern learning is done in state (2) to allow more general patterns to be learned. Applying the patterns in state (1) means that some of the learned contexts will not be applicable (i.e. those containing `&NAME`, `&NUM` or `&CHAR`). However, most of the greeting and closing patterns do not rely on such contexts, and application of the patterns in this state enables entity names to be learned that would otherwise be missed. Clearly though, we must also apply the patterns in state (2) so that those reliant on `&NAME` etc. context can take effect.

As the pattern application proceeds, all anonymised words are written to a log file. Aggregating the log file yields a list of distinct anonymised terms, which we manually examine[12], removing all terms that are not genuine entity names. This list of terms is then propagated through the data, yielding more tagged proper names and potentially altering the contextual properties of the data, allowing further application of pattern anonymisation. Again, the list of anonymised terms is examined and propagated. This process can be iterated as many times as is necessary, and if sufficient data is available the propagated terms can be used to learn new patterns.

The extent to which manual intervention is required depends on the amount of data being anonymised at a given time. For example, we can apply previously-learned patterns and lists of anonymised words to individual messages, as well as propagating newly-learned proper names without worrying too much about over-anonymisation. At the same time, though, we are unlikely to learn many useful new instances. However, if we are anonymising a large amount of data, then we must supervise the pattern generation and construction of anonymised word lists so that spurious instances are not widely propagated.

3.2.6 Manual Pattern Anonymisation

The final phase of the anonymisation procedure is to apply manually-constructed patterns, drawing on the general structure of email data. For example, if a line contains just a single word, the within-sentence linguistic context cannot reliably suggest which class that word should belong to (if it does not occur in the lexicon). *Note that this differs from the word-comma case observed earlier*. However, we know from the structure of email (as well as mail in general) that if such a line occurs at the beginning of the message body, it is highly likely to be either a greeting term or a proper name.

We also manually construct patterns for contexts that are known to be highly suggestive of proper names but have not occurred a sufficient number of times in the data to be learned automatically. For example, phrases such as:

‘`Rev. Plumb`’ and ‘`Messrs. White and Black`’

are recognised by the following patterns:

```
LC<\S+_NNSB> RC<>

LC<\S+_NNSB2 &NAME_NP1 \S+_CC> RC<>
```

Such patterns are added to the list of those automatically learned earlier. Once more, the newly-learned proper names can be propagated and the whole process iterated as necessary.

[12] An alternative method of validation could be devised, using for instance an online dictionary.

3.3 Experiment

In order to demonstrate the anonymisation procedure we drew 50 messages at random from our genuine email corpus and applied the various phases discussed above, examining the effects on the test set at each stage.

The email data we used for this experiment consisted in total of approximately 11,000 words of text, drawn from both the message body and subject fields. The first step was to PoS tag the data. The tagger found 1076 proper names in total, 469 of them distinct. Without a time-consuming analysis it is difficult to report exactly the precision of the tagger in this instance, but after examining the list of proper names identified, we estimate it at around 80%. Whilst it is clearly desirable to reduce the number of words misclassified as proper names, we suspect that it only marginally affects the overall classification task, if at all.

We then applied the patterns learned from the whole corpus to the unanonymised data, which yielded 17 proper names with 100% precision. Applying the same patterns to the anonymised data resulted in a further 4 proper names, although the veracity of one was debatable. Neither the manual patterns nor subsequent iterations of the learned patterns yielded any further matches.

We then examined the resulting anonymised messages and discovered 10 distinct proper names that had been missed. However, propagating names learned in other parts of the corpus, we were left with just 4 of the 10. The 4 remaining proper names consisted of one acronym for a university course, one abbreviated first name (recognised as a foreign word), one foreign name (given in lower case with very little context) and one place name. It is possible that there were other proper names that we missed when examining the anonymised text, but these figures give some impression of the effectiveness of the approach.

The results are summarised in the following table:

Data set consisting of 50 email messages, ~ 11,000 words		
Anonymisation Phase	**Number of Proper Names Identified**	**Correctly**
PoS tagging	1076	~ 900
Learned pattern application (1)	17	17
Learned pattern application (2)	4	3
Manual pattern application	0	0
Further LPA iterations	0	0
Corpus-Wide Learned PN's	6	6
Missed Instances	4	-

One issue raised by this experiment is that of whether phrases whose individual words may not be proper names, such as '*Department of Geological Sciences*', should be anonymised. We suggest that such phrases are sufficiently vague as to be insensitive, presuming that any terms relating them to a specific location/organisation have themselves been anonymised.

3.4 Discussion

3.4.1 Comparison

Our approach to anonymisation is a domain-specific hybrid PoS tagging and semi-supervised ML technique, using linguistic knowledge and bootstrapping to learn patterns. It is really a simplified form of the more general problem of named entity recognition (NER). We have already mentioned the work of Mikheev, who has contributed a number of quite successful approaches to NER, such as the system that won the MUC-7 (*Message Understanding Conference*) NER track, based on a cascading rule-based and probabilistic techniques (Mikheev et al. 1998). More recently, Agichtein and Gravano (2001) propose an iterative bootstrapping approach to pattern learning that generalises to various forms of information extraction. It is somewhat similar to our pattern learning approach, though significantly more complex, implementing an involved functional model for identifying reliable patterns.

3.4.2 Improvement

The procedure could be improved in a number of ways. Above all, the nature of the task requires high recall, and this could be improved by using more sophisticated pattern-learning techniques (e.g. Agichtein and Gravano 2001) and by training the tagger on domain-specific annotated data. Precision is also desirable, and the misidentification of common words as proper names within the tagger could again be improved by training on domain-specific data and by improving the analysis of unknown words. Finally, it would be useful to do full named entity recognition to distinguish between, e.g. person and place names, rather than associating every proper name with the same token.

Ch 4. LM Classification Model

We now present and discuss our generative language model classification approach, covering both the formal classification model as well as such issues as LM construction and estimation of priors.

4.1 Introduction

Statistical language models have been widely used in the field of speech recognition to assign probabilities to arbitrary word sequences in a given language. The probabilities are estimated from data, using smoothing techniques such as *discounting* and *backing-off* to compensate for sparsity.

More recently, a number of researchers have shown that language modelling techniques can be successfully applied to *information retrieval* (Ponte and Croft 1998, Hiemstra and de Vries 2000), although the techniques they propose tend to be only rather loosely based on traditional language modelling ideas.

The technique we present differs from extant text classification methods although it shares something in common with generative strategies such as Naïve Bayes. It also differs significantly from the LM approaches to IR described by Hiemstra and de Vries (2000) and others, though some of the basic framework is adapted from these ideas.

Although in this project we only consider application of the proposed LM classification technique to the 2-class spam filtering problem, it is also applicable to the more general N-class text classification problem, as is made clear in the presentation of the formal model.

4.2 Formal Model

4.2.1 Terminology and Assumptions

We adopt the following general terminology and definitions:

- *Document*: a domain-specific, discrete item of information (i.e. a single news-story, email, etc.). Documents are comprised of *words* and meta-level structures, such as *fields*.
- *Word*: an atomic unit within a document (note that this is broader than the linguistic notion).
- *Field*: a (possibly recursive) meta-level structure consisting of a label, and a finite number of words (and sub-fields).
- *Class*: a predefined (possibly infinite) set of documents, usually homogeneous with respect to a specified aspect of the information they contain (i.e. *topic*).
- *Classification*: a mapping of a given document to a given class.

We also make the following assumptions:

1) We assume that a document belongs to exactly one class, though the model can quite easily be extended to account for documents belonging to arbitrarily many classes.

2) We assume that a document is structured into exactly L fields, and that the fields are non-recursive, consisting only of words, as illustrated by the following diagram:

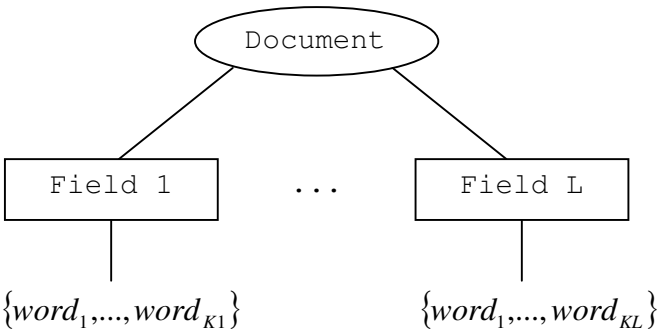

Figure 4.1: Document structure

3) We assume that the fields within a document are *independent* of each other.
4) Finally, we assume that classification is carried out within a single domain, and within that domain, all documents have the same structure.

4.2.2 Classification Framework

Given a set of classes $\{C_1,...,C_N\}$, and a set of documents $\{D_1,...,D_M\}$, we seek to discover a set of classifications $\{(D_1 \to C_{j_1}),...,(D_M \to C_{j_M})\}$ where j_i ranges from 1…N (given assumption 1).

In the generative framework, we classify a document as belonging to the class with the highest *posterior probability*:

$$\text{Decide}(D_i \to C_j) \text{ where } j = \arg\max_k [P(C_k \mid D_i)] \quad (1)$$

We will build on this model, eventually deriving an amended decision rule.

Using assumptions 2, 3 and 4 about documents, we define the posterior probability in terms of a weighted *interpolation* of the posterior probabilities of the *L* fields making up documents in this domain:

$$P(C_j \mid D_i) = \sum_{f=1}^{L} \lambda_f \left[P(F_{fC_j} \mid F_{fD_i})\right] \quad \text{such that} \quad \sum_{f=1}^{L} \lambda_f = 1$$

where F_{fC} refers to field number f in class C
F_{fD} refers to field number f in document D
λ_f is the weight given to field f

An interpolation scheme is used to find the optimal values for the λ's (this is discussed further in the results section).

We expand the posterior probability formula using *Bayes Rule*:

$$P(C_j \mid D_i) = \sum_{f=1}^{L} \lambda_f \left[\frac{P(F_{fC_j}) \bullet P(F_{fD_i} \mid F_{fC_j})}{P(F_{fD_i})} \right]$$

$P(F_{fC_j})$ is the *prior probability* for class C_j, representing the probability of a random document belonging to class C_j regardless of its content (see 4.4). It is assumed to be constant across all fields within a given class. $P(F_{fD_i} \mid F_{fC_j})$ is the *class-conditional probability*, and is calculated by the LM component of the classification model, discussed in the next section. Although $P(F_{fD_i})$ is constant with respect to class, it is useful under interpolation because it *normalises* the values given by fields with varying lengths. It can be expanded as the sum of the prior times the class-conditional probability for the given field across all classes:

$$P(C_j \mid D_i) = \sum_{f=1}^{L} \lambda_f \left[\frac{P(F_{fC_j}) \bullet P(F_{fD_i} \mid F_{fC_j})}{\sum_{k=1}^{N} P(F_{fC_k}) \bullet P(F_{fD_i} \mid F_{fC_k})} \right]$$

Dividing by $P(F_{fD})$ can be seen as scaling the class-conditional probability of a field (which is dependent on the length of the field by virtue of the fact that probabilistic intersection equates to multiplication) by a value constant with respect to class but multiplicatively proportional to the length of the field, hence normalising the field length whilst retaining the class-conditional proportionality.

Due to the diminishing nature of the values in a multiplicative probabilistic setting, we prefer to work with log probabilities rather than true probabilities to avoid computational underflow, and the monotonicity of the log function makes it an appropriate conversion:

$$score(C_j, D_i) = \sum_{f=1}^{L} \lambda_f \left[\log P(F_{fC_j}) + \log P(F_{fD_i} \mid F_{fC_j}) - \log\left(\sum_{k=1}^{N} P(F_{fC_k}) \bullet P(F_{fD_i} \mid F_{fC_k}) \right) \right] \qquad (2)$$

Note that we are now using the notion of a 'score' as the formula no longer returns a true probability. The final term in this formula cannot be directly converted to log probabilities, but we can algebraically manipulate it into a suitable form using an adaptation of the *Log-Add* principle[13]. We will consider the 2-class problem as an example:

[13] A commonly-used approximation when implementing log addition, due to its efficiency

$$\text{let } \alpha_k = P(F_{fC_k}) \bullet P(F_{fD_i} \mid F_{fC_k}) \quad \text{thus} \quad \log \alpha_k = \log P(F_{fC_k}) + \log P(F_{fD_i} \mid F_{fC_k})$$

$$\log(\alpha_1 + \alpha_2) = \log(\exp(\log \alpha_1) + \exp(\log \alpha_2))$$

$$= \log(\exp(\log \alpha_1)) + \log(\exp(\log \alpha_1) + \exp(\log \alpha_2)) - \log(\exp(\log \alpha_1))$$

$$= \log \alpha_1 + \log\left(\frac{\exp(\log \alpha_1) + \exp(\log \alpha_2)}{\exp(\log \alpha_1)}\right)$$

$$= \log \alpha_1 + \log(1 + \exp(\log \alpha_2 - \log \alpha_1))$$

Now, if we ensure that $\log \alpha_2$ is the smaller value, as the difference between $\log \alpha_1$ and $\log \alpha_2$ increases the second term in the expression approximates to zero. If the difference is small, then it is computationally feasible to calculate $\exp(\log \alpha_2 - \log \alpha_1)$. Thus, we set a threshold of feasibility for computation of the second term and assume otherwise that it is zero.

Finally, we rewrite the decision rule (1) such that it derives classifications from the document/class score function (2) rather than directly from the class posterior probabilities:

$$\begin{aligned} &\text{Decide}(D_i \rightarrow C_j) \\ &\text{where } j = \arg\max_k [score(C_k, D_i)] \end{aligned} \tag{3}$$

4.2.3 LM Component

The class-conditional score function (2) requires an estimate of the probability of a document field given a class field. In our model we take a document field to be a sequence of words, and a class field to be a *language model* built from the relevant fields of the documents known to belong to that class. Thus we have:

$$P(F_D \mid F_C) = P(w_{1F_D}, ..., w_{KF_D} \mid LM_{F_C}) \tag{4}$$

At this point, it will be helpful to introduce some traditional language model theory. A language model is used to estimate the probability that a given word sequence exists in the language modelled. Perhaps the most widely used type of LM is the *N-gram* model, where the probability of a word existing at a given position in a sequence is assumed to be dependent only on the previous *N-1* words, giving the probability of a word sequence as:

$$P_{N-gram}(w_1, ..., w_K) = P(w_1, ..., w_{N-1}) \prod_{i=N}^{K} P(w_i \mid w_{i-N+1}, ..., w_{i-1})$$

For example, in the *bigram* (2-gram) case:

$$P_{bigram}(w_1,...,w_K) = P(w_1)\prod_{i=2}^{K} P(w_i \mid w_{i-1})$$

Note that in the *unigram* (1-gram) case, the formula becomes a simple product of word probabilities and all ordering constraints are lost:

$$P_{unigram}(w_1,...,w_K) = \prod_{i=1}^{K} P(w_i)$$

There are many different types of language model, some attempting to provide a deeper analysis of language syntax and semantics by incorporating word-class clustering, non-adjacent relations, linguistic knowledge, high-dimensional semantic reduction, and various other techniques. The advantages gained by the use of such techniques come at the cost of (sometimes dramatically) increased computational complexity.

Within the proposed classification framework, language model type is actually unimportant. LM's are constructed to represent each field of each class, thus yielding log probabilities for the scoring function (2), which in turn yields results for the overall decision rule (3).

4.2.4 Incorporating Adaptivity

A classification model for spam filtering must include an adaptive element, enabling it to cope with varying definitions of what constitutes spam. In light of this, we extend our classification model at a high level by incorporating an interpolation of scores from both a static and dynamic element.

We expand the decision rule (3):

$$\begin{aligned} &\text{Decide}(D_i \rightarrow C_j) \\ &\text{where } j = \arg\max_k \left[\lambda_s score_s(C_k, D_i) + \lambda_d score_d(C_k, D_i)\right] \end{aligned} \tag{5}$$

The subscripts *s* and *d* denote the static and dynamic elements respectively, and a standard interpolation scheme is employed such that:

$$\lambda_s + \lambda_d = 1$$

This allows us to weight the contribution from the static and dynamic elements. The idea behind the modified decision rule is that the static element represents a classification model built from a large background corpus, while the dynamic element represents a much smaller model that is regularly retrained on newly-classified data.

Of course, it is always possible to retrain the static element when enough new data has been accumulated.

4.3 LM Construction

In this section we discuss the construction of various language models with their relevant parameters, and present two variants of what is to our knowledge a new *discounting* function for N-grams, based on analysis of using LM's for classification within the proposed framework.

4.3.1 Unigrams

The simplest type of language model is the unigram. Recall the formula from the previous section:

$$P_{unigram}(w_1,...,w_K) = \prod_{i=1}^{K} P(w_i)$$

Thus, for each word, w_i we require an estimate of its probability. A common method of estimating these probabilities is to use maximium likelihood estimation (MLE), i.e. given some training data, we calculate probabilities for the words that maximise the likelihood of the language model generating that data. In the unigram case, the ML estimate of the probability of a word is:

$$P(w_i) = \frac{freq(w_i)}{\sum_{j=1}^{V} freq(w_j)}$$

where $freq(w)$ denotes the number of times word w occurs in the training data
V denotes the vocabulary size

Based on this formula, unseen words are assigned a probability of zero, which is undesirable, so we use a *smoothing* technique to assign some of the probability mass to unseen events. Unless we are working within a tightly constrained vocabulary domain it is impossible to strictly uphold the probability model when smoothing, because we are unable to correctly estimate the size of the vocabulary (e.g. in English it is always changing). Thus we must make the simplifying assumption that we know the vocabulary size. There are various smoothing techniques, but we use a method whereby some small probability (known as a *floor*) is specified for unknown words. This floor is then multiplied by the estimated number of unseen words, and that portion of the probability mass is reserved. The rest is assigned in an ML fashion to the words in the training data. This approach is represented by the following formula:

$$P(w_i) = \begin{cases} (1 - U\alpha F)\dfrac{freq(w_i)}{T} & \text{if } freq(w_i) > 0 \\ F & \text{otherwise} \end{cases}$$

where F = *floor* probability
α = *floor* weight
T = number of words in training data
U = estimated number of unseen words

The advantage of this approach is that *F* can be directly estimated to model the probability of encountering an unknown word, given a class. The *floor weight* is useful in situations where *F* cannot easily be estimated directly (i.e. in an adaptive setting). Then, we use α to model the *relative* probability of an unknown word given a class, where *F* is set to an LM-dependent value such as 0.5/*T*. Otherwise, α defaults to 1. In practice we have found that the value of *U* can be assumed irrelevant unless for some reason a large proportion of the probability mass is to be assigned to unseen words.

4.3.2 Bigrams

When constructing bigram language models, we estimate the probability of one word w_j occuring after another w_i, or formally $P(w_j \mid w_i)$. The ML estimate is as follows:

$$P(w_j \mid w_i) = \frac{freq(w_i, w_j)}{\sum_{k=1}^{V} freq(w_i, w_k)} = \frac{freq(w_i, w_j)}{freq(w_i)}$$

Data sparsity is a more serious problem here than in the unigram case, as there are now V^2 potential bigrams, many of which are unlikely to have been seen a reliable number of times without a very large training data set. For example, given a vocabulary size of 50k, there are 2500m possible bigrams. If we estimate that around a quarter of these might reasonably be expected to occur, and that we require on average somewhere in the region of 5 instances per bigram for reliable statistics, then we would need a training set of around 3 billion words. Such training sets are available for certain domains, but in general much smaller sets must be used.

Various techniques have been suggested for handling the sparsity problem when using N-grams. Two popular approaches are *deleted interpolation*, introduced by Jelinek and Mercer (1980) and *backing off*, introduced by Katz (1987). We use a backing off strategy, as it is somewhat easier to implement and often more efficient[14], although relative performance of the two techniques varies with differing properties of the models and the training data sets used (see Chen and Goodman (1996) for comparison).

The idea is that without reliable information about the bigram probability of a word, we *back off* to its unigram probability. This process consists of two stages: first we must *discount* some of the probability mass from seen bigrams, and then we must assign it to unseen bigrams based on the unigram distribution. The general formula for backed-off ML bigram estimation is as follows:

$$P(w_j \mid w_i) = \begin{cases} d(freq(w_i, w_j)) \dfrac{freq(w_i, w_j)}{freq(w_i)} & \text{if } freq(w_i, w_j) > C \\ \alpha(w_i) P(w_j) & \text{otherwise} \end{cases} \qquad (6)$$

where $d(r)$ is the *discounting function*

$\alpha(w)$ is the *back - off weight* for unigram word *w*

C is the *N - gram cut - off point*

[14] For details of the deleted interpolation smoothing strategy as well as various other related issues, see any text on automatic speech recognition (ASR), for instance, Huang et al. (2001).

The formula is due to Katz (1987) who first suggested combining higher and lower-order N-gram distributions by discounting and backing off. The back-off weight for word w_i is the total amount of probability mass that was discounted from bigrams of the form $(w_i,*)$. This is then distributed according to the unigram probabilities as shown in the formula. If the back-off weight for w_i is chosen correctly, then the following will hold:

$$\sum_{j=1}^{V} P(w_j \mid w_i) = 1$$

Actually, this holds only under the assumption that we know the vocabulary size, which is untrue, but as for the unigram case this assumption is unproblematic in practice.

The N-gram cut-off point, *C*, can be seen as the threshold below which we consider the number of occurrences too low to draw robust statistics from. For instance, if *C* is set to 2, then all bigrams that occur three or more times will be included in the model. In some cases, we may want to raise the value of *C*, as low-frequency bigrams may be less reliable than their unigram counterparts.

The value of *C* can also be used to control the size of the language model – the higher it is, the smaller the resulting LM will be.

4.3.3 Discounting

This leaves us to discuss the discounting function. Various discounting schemes have been suggested, such as *absolute*, *linear*, *Good-Turing* and *Witten-Bell*. We implemented linear and Good-Turing approaches for the project, as well as defining a new approach, and will consider these three in detail. For information on other methods, see an ASR text, such as Huang et al (2001).

The *linear* discounting function usually takes the following form:

$$d(r) = 1 - \frac{n_1}{T}$$

where n_r = the number of bigrams that occurred *r* times

T = the total number of bigrams seen

This is equivalent to multiplying each bigram probability by a fractional constant, thus discounting a quantity of the probability mass linearly proportional to the probability of each bigram. This scheme is quite effective as a general discounting measure, but highlights a potential weakness in the N-gram model when utilised for classification tasks. The weakness stems from the fact that assuming a word has been seen more than the cut-off number of times (*C*) in a certain context, its probability is dependent only on the relative number of other words also seen in that context.

For example, consider the two-word phrase "*happy birthday*". Given two sets of training data, T_1 and T_2, assume that "*happy birthday*" occurs three times in T_1 and fifteen times in T_2, and that there are no other occurrences of the word "*happy*" in either. Before discounting, the language models built from T_1 and T_2

(LM_1 and LM_2) would both assign $P(birthday \mid happy) = 1.0$, regardless of the fact that the bigram is more frequent in training set T_2 than in set T_1, and thus more likely to have been generated by LM_2 in a comparison, assuming that T_1 and T_2 are of similar overall size. A similar argument could be made using two different bigrams within the same language model. The underlying problem is that within the N-gram framework we don't explicitly model our 'confidence' in a greater-than-one-gram probability estimate (as long as it has been seen more than *C* times).

It turns out that discounting can be used not only to handle unseen N-grams, but also to provide some account for this potential weakness in N-gram modelling by discounting probability mass in a manner inversely proportional to N-gram confidence (i.e. the number of times it has been seen). Linear discounting, however, will simply deduct the same quantity from each probability.

Good-Turing discounting is a technique that does take N-gram confidence into account. It is defined as:

$$d(r) = \frac{(r+1) \bullet n_{r+1}}{r \bullet n_r}$$

We can conceptualise the behaviour of this function by considering a plot of N-gram confidence against the number of N-gram instances seen in training data. The following graph was generated using data from the *GenSpam* corpus:

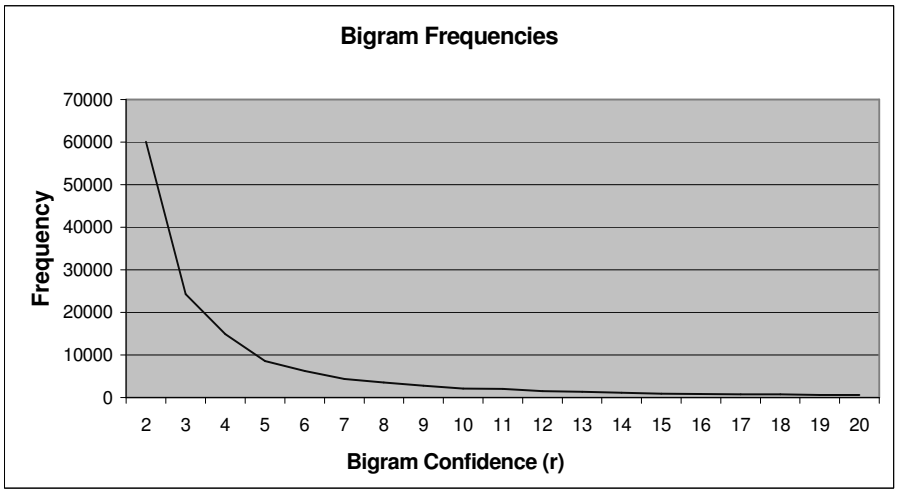

Figure 4.2: Bigram Confidence Frequencies

The distribution is approximately *Zipfian*, with the majority of N-grams occuring fewer than ten times, and the tail stretching far beyond the extremities of the graph shown. Katz (1987) suggests that it only really makes sense to use Good-Turing discounting for the relatively low N-gram frequencies, e.g. < around 7.

By plotting the results of the Good-Turing discounting function on the above values, we can observe its behaviour (we also plot the linear discounting function for comparison):

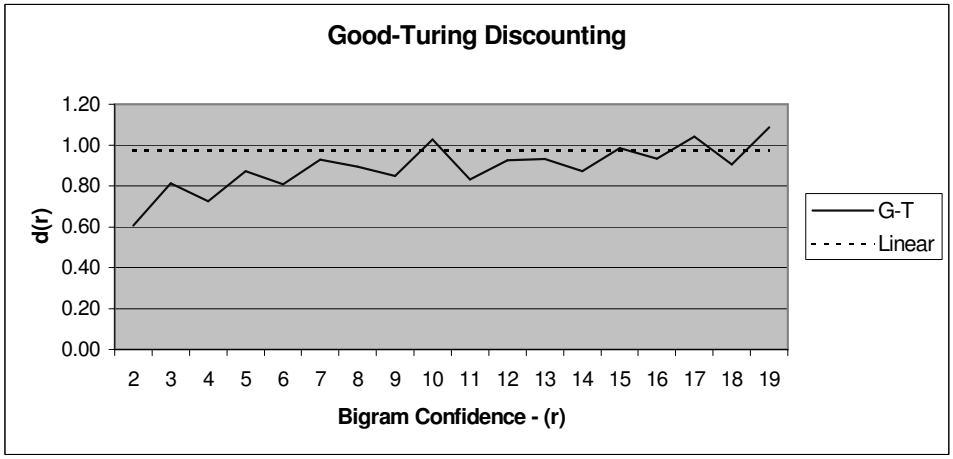

Figure 4.3: Good-Turing Discounting

As the distribution begins to flatten, the function becomes less well-behaved, sometimes returning a value in excess of 1.0. In practice Good-Turing is often used in conjunction with another technique to ensure well-behavedness. *Katz smoothing* (Katz 1987) uses Good-Turing discounting in the context of the general back off formula (6). It is perhaps the most widely used smoothing technique in the speech recognition domain. Various enhancements to Katz smoothing have been suggested, for instance Church and Gale (1991) propose a method using a technique called *bucketing* to cluster N-grams into disjoint groups described by related parameter sets. However, recent studies (Chen and Goodman 1996) suggest that this only improves upon Katz smoothing when trained on very large data sets, and otherwise tends to underperform.

From the graph it can be seen that the behaviour of the Good-Turing function is somewhat erratic even on the lower frequency N-grams – for example, there seems to be no reason to discount bigrams that occur 4 times more aggressively than those that occur 3 times.

Bearing this in mind, we propose a new discounting technique, which we will call *confidence discounting*, defined as:

$$d(r) = \frac{r(R-1)}{\frac{R}{\omega}(r-1) + \frac{1}{\phi}(R-r)}$$

where R = the number of distinct frequencies

ϕ = floor for lowest confidence

ω = ceiling for highest confidence

The idea is that the value of the discounting function represents the confidence we have in the N-gram, based on its count, restricted to fall between the values ϕ and ω. R is an estimate of the highest level of confidence, chosen as the number of distinct N-gram frequencies because of its robustness to outliers. ϕ is chosen to represent the quantity of probability mass retained in the case of least confidence (i.e. when the N-gram count is 1), and ω is chosen to represent the quantity of probability mass retained in the case of highest confidence (i.e. when the N-gram count approaches R). The optimal values for ϕ and ω can be estimated through empirical analysis, although setting them as follows will cause the

function to approximate Good-Turing discounting for lesser values of *r* and linear discounting as *r* approaches *R*:

$$\omega \approx 0.5 \quad , \quad \phi = 1 - \frac{n_1}{T}$$

The following graph illustrates the behaviour of the confidence discounting function:

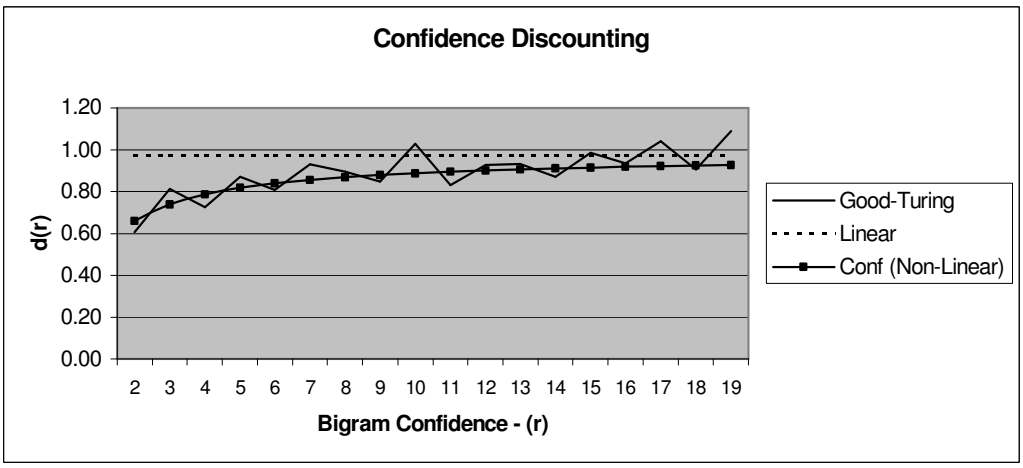

Figure 4.4: Confidence Discounting

We also tested an alternative linear version of the confidence discounting function, defined as follows:

$$d(r) = \frac{r}{R}(\omega - \phi) + \phi$$

The function, plotted over a larger range of *r* appears as follows:

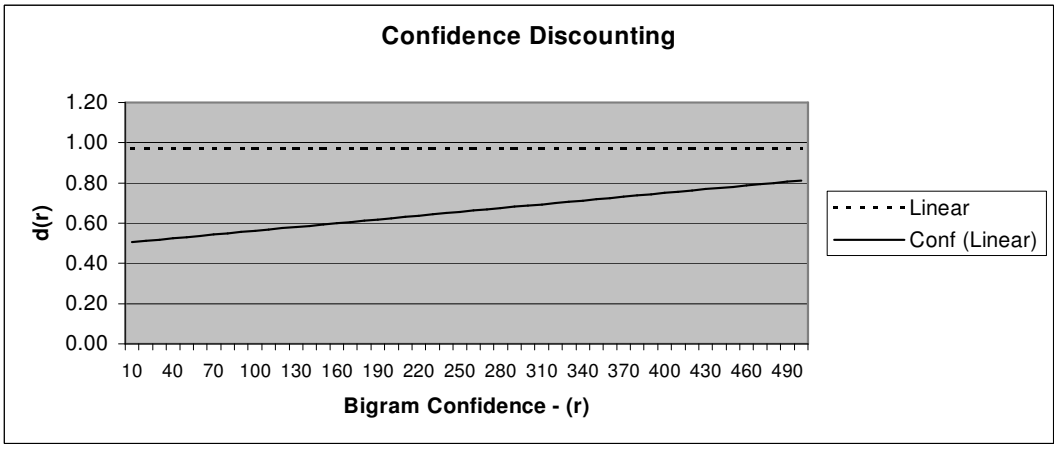

Figure 4.5: Confidence Discounting

The assumption behind the non-linear form of the function is that confidence in a given N-gram should increase significantly after it occurs the first few times, and then continue to increase at a slower rate as it is encountered more and more often. The justification for this assumption is that as if an N-gram has been seen more than a few times (say around 5-10) it is likely to be more than just an erroneous or exceptional

case, and our confidence in it should increase rapidly. However, once an N-gram has been seen multiple times already, seeing it a few more times should not imply such a significant increase in confidence.

The assumption behind the linear form of the function is that if N-gram g_1 has been seen twice as often as g_2, our confidence in g_1 should be twice as great as our confidence in g_2, regardless of the relative number of times both have been seen.

In the results section, we consider the performance of the various discounting schemes in practice.

4.3.4 Higher-Order N-Grams

The issues faced when constructing higher-order N-grams are actually very similar to those encountered in the construction of bigrams. However, instead of having a single back-off operation (from bigram to unigram), we have a *back-off chain*, illustrated by the following diagram for the *trigram* case:

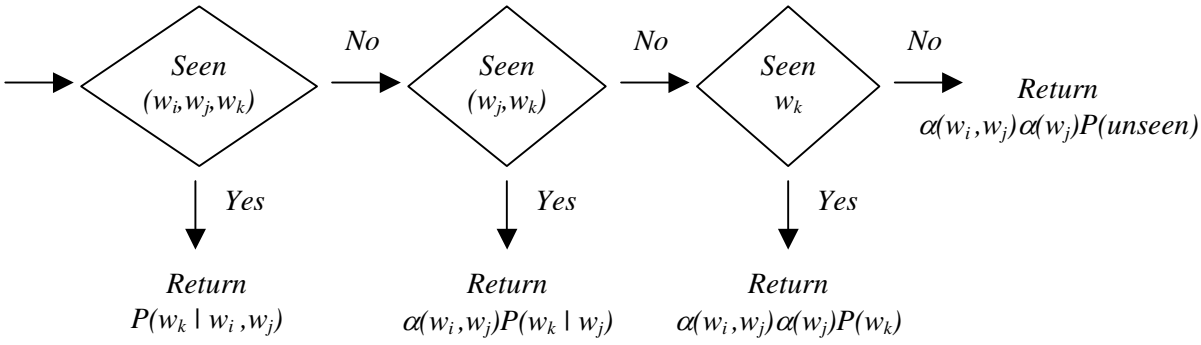

Figure 4.6: Trigram back-off chain

Note that similarly to the unigram back-off weight function, $\alpha(w_i, w_j)$ is calculated to ensure that:

$$\sum_{k=1}^{V} P(w_k \mid w_i, w_j) = 1$$

An issue that arises particularly with the use of higher-order N-grams is that of *overfitting*. As N grows, the training data is modelled with increasing precision; however, if there is a lack of training data or it fails in some way to represent the general domain, the model will not *generalise* well to unseen data. Smoothing techniques help to overcome this, but it is still a significant problem unless very large quantities of representative training data are available.

4.4 Estimating Priors

The scoring function (2) requires an estimate of the prior probability for a given class. Such estimates should reflect our *a-priori* knowledge of the relative sizes of the classes in the classification domain. For development purposes we usually estimate priors based on the size of the training/test data sets. In the application of a statistical solution to the spam filtering problem the priors might be estimated based on the frequency of received genuine email compared to that of spam for a particular user.

Ch 5. Results and Analysis

In this section we describe our experimental method, and present results obtained on the *GenSpam* and *LingSpam* corpora using the proposed classification model with various LM types and parameter settings.

5.1 Evaluation Measures

In our experiments we report *precision* (*p*), *recall* (*r*) and *F1* for both classes – SPAM and GEN. These measures are defined as follows:

$$p(C) = \frac{num(D_C \rightarrow C)}{num(* \rightarrow C)} \quad r(C) = \frac{num(D_C \rightarrow C)}{num(D_C)} \quad F1(C) = \frac{2 \times p(C) \times r(C)}{p(C) + r(C)}$$

where $num(D_X \rightarrow Y) = \text{the number of documents of class } X \text{ classified as } Y$

$num(D_X) = \text{the total number of documents of class } X$

There is some debate as to whether it is useful to report both recall and precision for the 2-class classification problem, as recall for one class is strongly correlated with precision for the other, especially if the test sets are of similar size. Indeed, it can be argued that it is enough to know either the recall for each of the two classes or the recall and precision for one; however, we will report all three measures for consistency across the classification field and to aid those accustomed to working with such values.

It has been argued by various researchers that for the spam filtering problem it would be useful to provide a performance measure that reflects the fact that misclassifying a genuine email is significantly more serious than misclassifying a spam email. This perception stems from the view that the average user would probably be prepared to manually remove a few spam messages that elude the filter, but does not want his/her potentially important genuine email to be discarded as spam. In light of this, Androutsopoulos et al. (2000a) propose a weighted accuracy measure, defined as:

$$WAcc = \frac{\lambda \bullet num(D_{GEN} \rightarrow GEN) + num(D_{SPAM} \rightarrow SPAM)}{\lambda \bullet num(D_{GEN}) + num(D_{SPAM})}$$

This represents a scenario in which it is λ times more costly to misclassify a genuine message than to misclassify a spam message. The weakness of this measure is that the values it yields are unintuitively high, assuming a reasonably large number of genuine messages are classified correctly. For example, if we assume that λ=9, a value suggested by both Androutsopoulos et al. (2000a) and Carreras and Marquez (2001), the following is a typical scenario:

$$\left.\begin{aligned} p(GEN) &= 0.70 \\ r(GEN) &= 0.95 \\ p(SPAM) &= 0.92 \\ r(SPAM) &= 0.60 \end{aligned}\right\} \Rightarrow WAcc = 0.92$$

Here we have assumed test sets for GEN and SPAM of 1000 documents each, and arbitrarily chosen the number of correct classifications in each case. The values demonstrate that *WAcc* cannot be directly compared with the other measures, and indeed does not range in any meaningful way from 0 to 1. Rather, it must be compared against a *baseline value*, which is derived from the test document counts and is again somewhat unintuitive (0.90 for the above example).

Androutsopoulos et al. also incorporate the λ variable into their classifier decision rule, requiring that a message be λ times more likely to be spam before it is classified as such. This means that as λ increases, the likelihood of classifying a message as genuine also increases, thus reinforcing the *WAcc* score.

Bearing this in mind, we suggest that the weighted accuracy measure is not actually very useful in practice. This is due in part to its unintuitiveness and non-uniformity in comparison to the standard performance measures, and also to the fact that increasing λ mutually increases the corresponding *WAcc* score, resulting in a scenario for high values of λ in which classification performance on spam is virtually irrelevant to the *WAcc* score.

We should also consider how a spam filtering system would be used in practice. As noted in much of the spam filtering literature, it is unlikely that a message flagged as spam by the filter would immediately be deleted. Rather, it would be stored in a folder reserved for spam, and deleted at some point later (probably after it had been used for retraining the adaptive component of the filter). It certainly makes sense to allow some kind of margin to be specified below which a message will be retained as genuine even if it receives a higher spam score, but using very large margins (high values of λ) that result in perfect genuine classification scores (and thus very high *WAcc* scores) at the expense of spam classification performance is not particularly practical, especially when the system would most likely be used in conjunction with a *whitelist* approach that automatically accepts email from known sources.

It is unlikely that any useful filtering technique would be able to guarantee perfect genuine classification performance, so the user will probably always have to scan filtered email at intervals; however it is clearly important that the genuine classification performance is as good as can reasonably be expected. We might therefore consider specifying a value above which the error rate for genuine classification becomes unacceptable. The margin can then be adjusted to ensure that performance meets this constraint.

5.2 Data Sets

We ran experiments using the *LingSpam* and *GenSpam* corpora, with the data divided as follows:

5.2.1 GenSpam Data Sets

We randomly draw two sets of 1000 messages (500 genuine, 500 spam) from the *GenSpam* corpus (described in 3.1) to be used for initial evaluation of LM features (which we will call EV1 and EV2), and use the remaining messages in each case for training (TR1 and TR2):

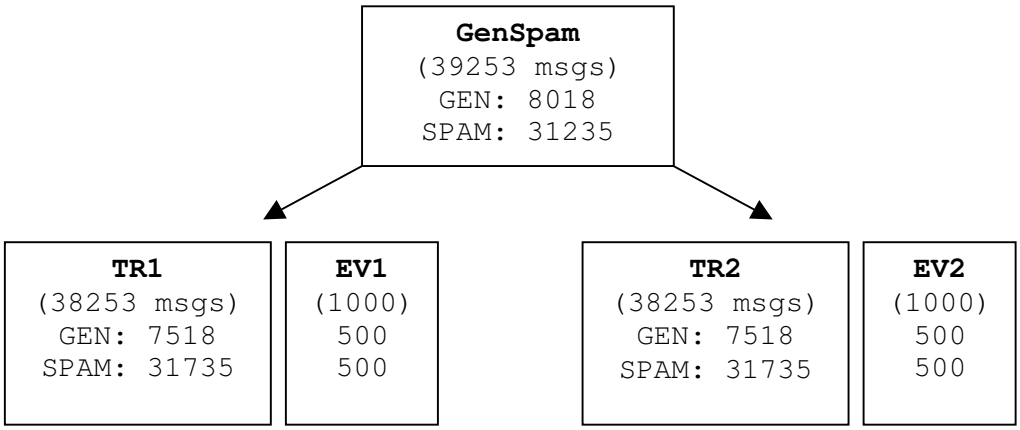

Figure 5.1: GenSpam Data Sets

Note that the disparity between the number of messages in the genuine and spam training sets is not a serious issue for the reasons discussed in 3.1.

We also draw a set of 6000 messages (3000 genuine, 3000 spam) from the GenSpam corpus for estimation of subject/body interpolation weights (see 5.3.1) which we will call WS and use the complement for training.

5.2.2 Unseen Test Set

For final experiments with the GenSpam corpus, we have assembled an unseen test data set consisting of:

- 1054 genuine messages
- 1097 spam messages

These messages were drawn from personal email sent and received (both genuine and spam) over the last year or so by ourselves and one other colleague, and thus represent a reasonable model of a typical user's inbox. We extract 600 messages (300 genuine, 300 spam) from the unseen test set to use as training for the adaptive component of the classifier (which we will call AD) and use the remaining messages for final testing (TS), as illustrated by the following diagram:

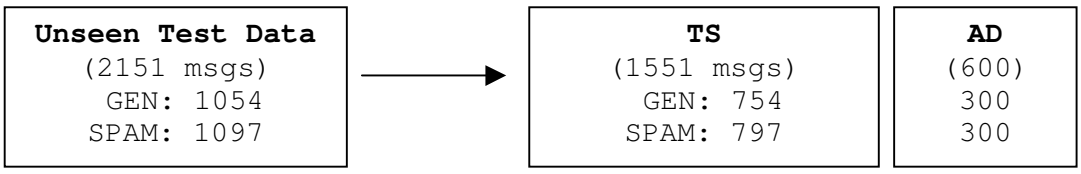

Figure 5.2: Unseen Data Sets

5.2.3 LingSpam Data Sets

The *LingSpam* corpus is divided into ten sections, which we use for ten-fold cross-validation, in line with the experiments carried out previously by Androutsopoulos et al. (2000) and others.

5.3 Experimental Method

The experiments carried out on the two corpora differed somewhat in their form. First we will discuss the concept of an *interpolation scheme*, and then we will consider the form of the specific experiments.

5.3.1 Interpolation

The purpose of an interpolation scheme is to optimise the weights of two or more interpolated components with respect to their performance on a given data set. A component in this sense is a function of a model or set of models trained on a given data set.

Usually the components being interpolated are both derived from the same data (e.g. lower and higher-order N-gram models), and optimality is with respect to maximising probability. In this case the training set must be divided into two disjoint sections, with the parameters for the statistical model estimated from one and the optimal interpolation weights estimated from the other. This is necessary because otherwise the weights simply converge to one extreme. In our case we are optimising with respect to classification performance, and thus both the statistical model and the interpolation weights could be estimated from the same data. However, this may not be preferable due to the fact that if the models are sufficiently complex to provide very high classification performance on the training data, the optimal weights will still tend to converge to extremes depending on which component has the closest fit. Thus, for our experiments we usually train and optimise weights on disjoint data sets.

To formalise the concept of an interpolation scheme, we will assume that the interpolation is between exactly 2 components, and define it as consisting of the following:

A, B	model components
$i(\lambda_A) = \lambda_A \cdot A + (1-\lambda_A) \cdot B$	*interpolation function* – interpolates A & B given a weight
$p(i, \lambda_A) = y$	*performance function* – returns performance value y given interpolation function and weight
$\lambda_1 ... \lambda_N$	ordered range of weights such that $\forall i \bullet 0 \le \lambda_i \le 1$
$o(p, i, \lambda_1 ... \lambda_N) = \arg\max_i [p(i, \lambda_i)]$	*optimisation function* – returns arg of optimal weight for A
$\lambda_A = \lambda_{o(p,i,\lambda_1 ... \lambda_N)}$	optimal weight for A
$\lambda_B = 1 - \lambda_{o(p,i,\lambda_1 ... \lambda_N)}$	optimal weight for B

When interpolating between document field scores in our classification model, the *interpolation function* is the scoring function (2), the *performance function* is the result of running the classifier (3) on the test set and calculating the performance measures, and the *optimisation function* applies the classifier to the test set with the specified range of weights to derive optimal values. We use this interpolation scheme in the experiments described below.

5.3.2 GenSpam Experiments

For the *GenSpam* experiments, we used the following method:

Phase 1:

(For unigram and bigram LM's):

1. Using model features θ_i:
   - For $N \in \{1,2\}$:
     - Train models on TR*N*.
     - Run classifier on EV*N*.
     - Calculate performance measures.
2. Adjust features $\theta_i \Rightarrow \theta_{i+1}$
3. Iterate $\Rightarrow$ 1 until sufficient feature variations tested.
4. Choose features $\hat{\theta}$ that maximise performance on EV1 and EV2.

Phase 2:

(Using model features $\hat{\theta}$)

- For each different LM type:
  - Train models on whole *GenSpam* corpus.
  - Estimate weights on WS.
  - Run classifier on unseen test set TS.
  - Calculate performance measures.
  - Train adaptive models on AD.
  - Estimate adaptive weights on AD.
  - (Re-estimate primary weights on AD).
  - Run classifier with adaptive component on TS.
  - Calculate adaptive performance measures.

Note that the different LM types include unigram, bigram (with 3 different discounting techniques) and trigram.

5.3.3 LingSpam Experiments

For the *LingSpam* experiments we used the following method:

1. Using model features θ_i:
   - For each of the ten sections:
     - Train body and subject models on remaining messages with parameters θ_i.
     - Use interpolation scheme on remaining messages to calculate weights.
     - Classify test section.
   - Calculate performance measures over all classified documents.
2. Adjust model features $\theta_i \Rightarrow \theta_{i+1}$.
3. Iterate $\Rightarrow$ 1.

Note that the adaptive component wasn't used for these experiments. This was partly due to the homogeneity of the data, and partly to provide results comparable with previous work.

We elaborated on the standard interpolation scheme for these experiments, due to the diminutive size of the *LingSpam* corpus (especially the spam component) and the fact that the sections are not divided evenly (some are significantly smaller than others). This results in a high level of optimal weight deviation between sections and makes the standard interpolation scheme unreliable. To counter this, we ran the ten-fold cross validation twice. On each of the ten initial iterations, we set aside once section of the data as the 'unseen' test set, and partitioned the others into a parameter estimation set (5 sections) and an interpolation weight estimation set (4 sections). We then calculated the *median* subject component weight over the ten iterations and re-ran the experiment using that value instead of re-calculating the weights on each iteration. The median was chosen for its robustness to outliers. The following diagram illustrates this approach:

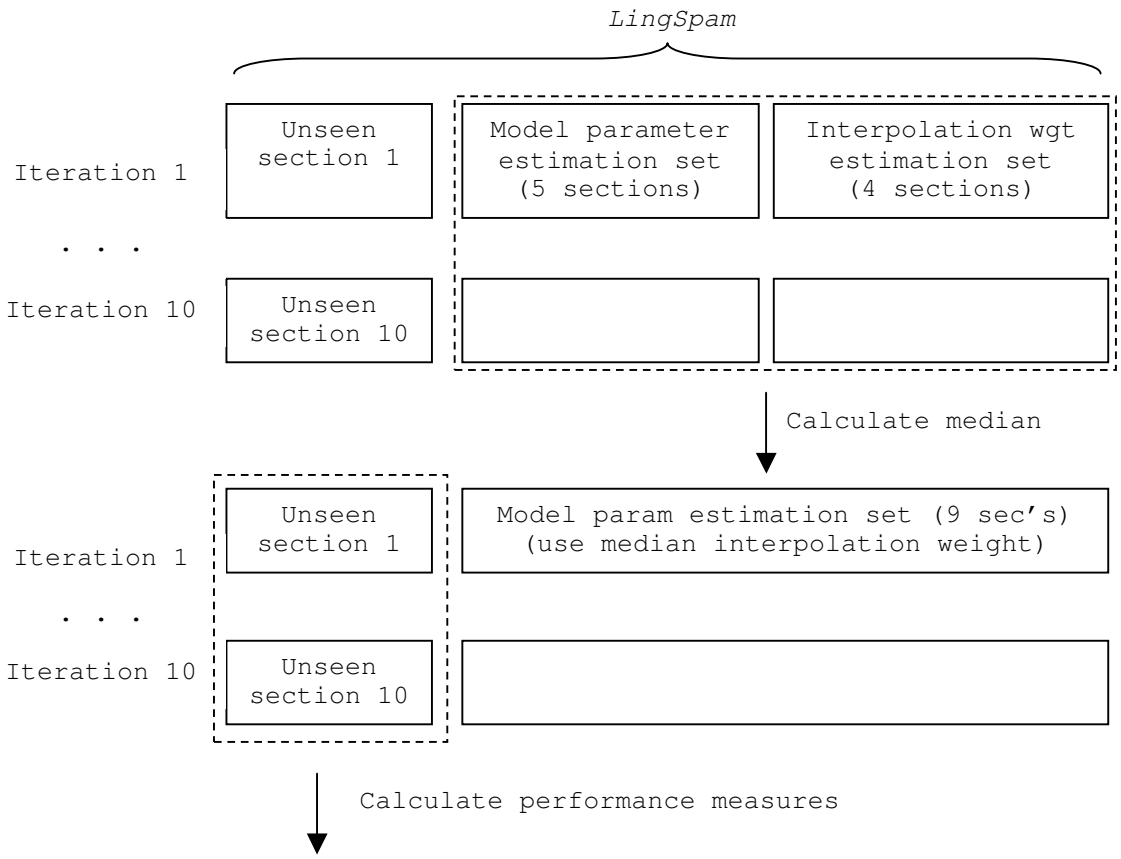

Figure 5.3: *LingSpam* experimental method

5.4 Results

5.4.1 GenSpam Results

Phase 1

During the phase 1 experiments (see 5.3.3) we varied certain features of the unigram and bigram LM's and observed results on held-back sections of the training data (EV1 and EV2) to determine the best-performing configurations. The features we considered were as follows (after each description is a key which will be used to represent the feature in subsequent graphs):

- Training a combined body and subject field LM and using it to model both fields. (*ALL*)
- Training separate LM's for the subject and body fields. (*SEP*)
- Removing single occurrences. (*SO*)
- Converting all N-gram occurrences to lower case. (*LC*)
- Setting the N-gram cut off point to *x* (bigram only). (C=*x*)

The unigram results for phase 1 were as follows:

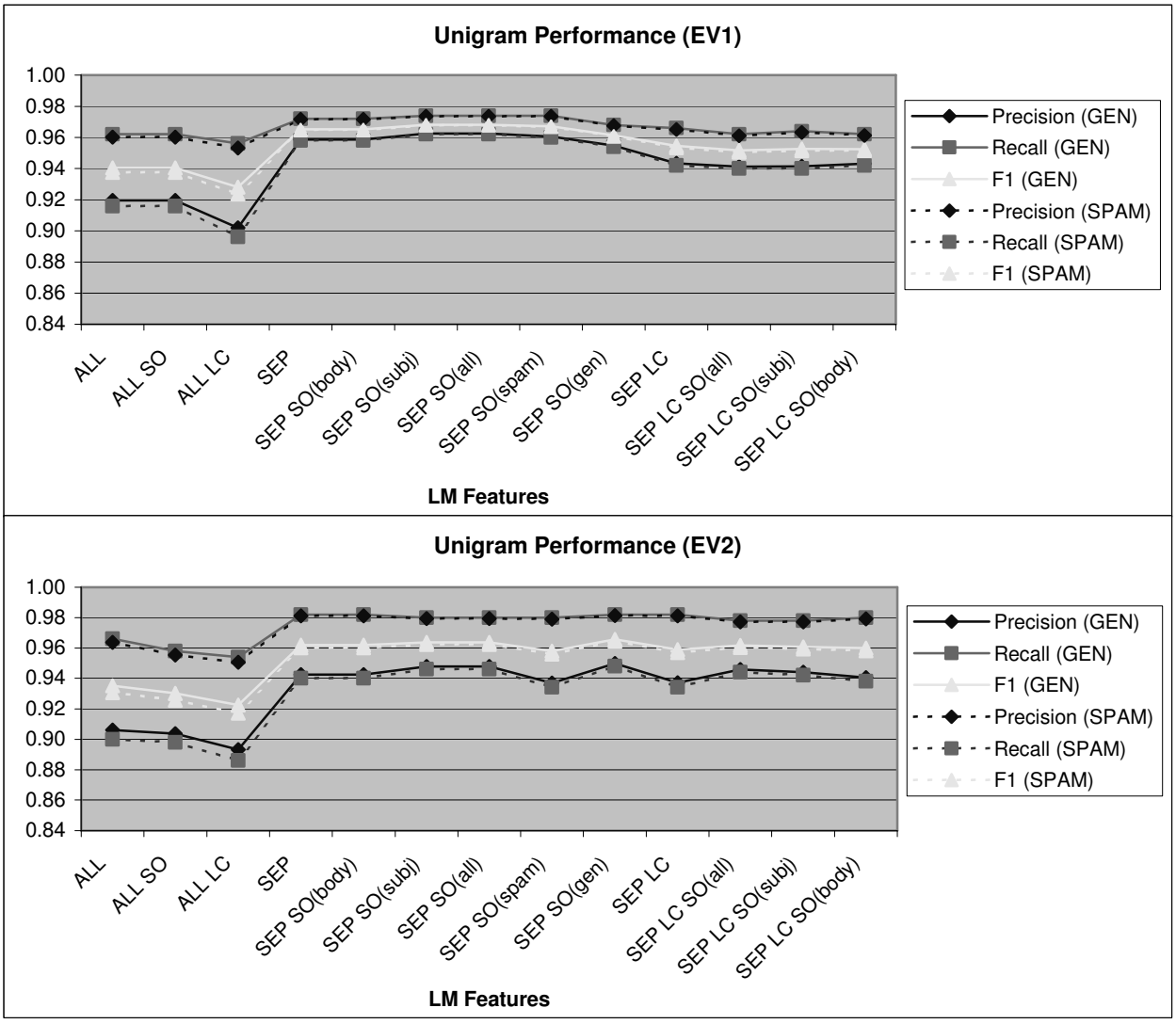

Figure 5.4: Phase 1 unigram results – EV1 and EV2

A key followed by a bracketed modifier denotes that the feature was applied that part of the data.

A number of observations can be made from these results:

- It is clearly beneficial to model the body and subject fields independently, rather than using a larger combined LM for both.

- Removing single occurrences in most cases provides a marginal increase in performance. This is as we would expect, bearing in mind that words appearing only once in the training data are unlikely to be representative of the language being modelled. Removing single occurrences is also beneficial for reducing the size of the LM, especially in the spam case, where many random strings of characters are introduced to try and confuse statistical filters.
- Converting to lower case has a slightly detrimental effect on performance on the EV1 set, and relatively little effect in the EV2 case. As above, converting to lower case reduces the size of the LM as it aggregates entries that are distinct only due to case differences. In one sense this aggregation can be seen as an advantage, i.e. in some contexts, case distinct forms of the same word are not meant to convey different semantics, especially in the email domain where capitalisation is often dropped: "*Hi John! How are you?*" is probably semantically equivalent to "*hi john! how are you?*". On the other hand, capitalisation is often used to convey intensity of meaning, for example: "*BUY VIAGRA NOW!!!!*" conveys a stronger sentiment that "*Buy Viagra now!!!!*". Given these examples, a sensible approach might be to selectively convert to lower case (e.g. convert all words with only a single upper-case letter); however we didn't pursue that avenue of research.

From these results, we took the *SEP SO(all)* feature set to be the overall best-performing. Note that we are not interested in the absolute performance at this stage, only at the relative effectiveness of the different features.

The bigram results for phase 1 were as follows:

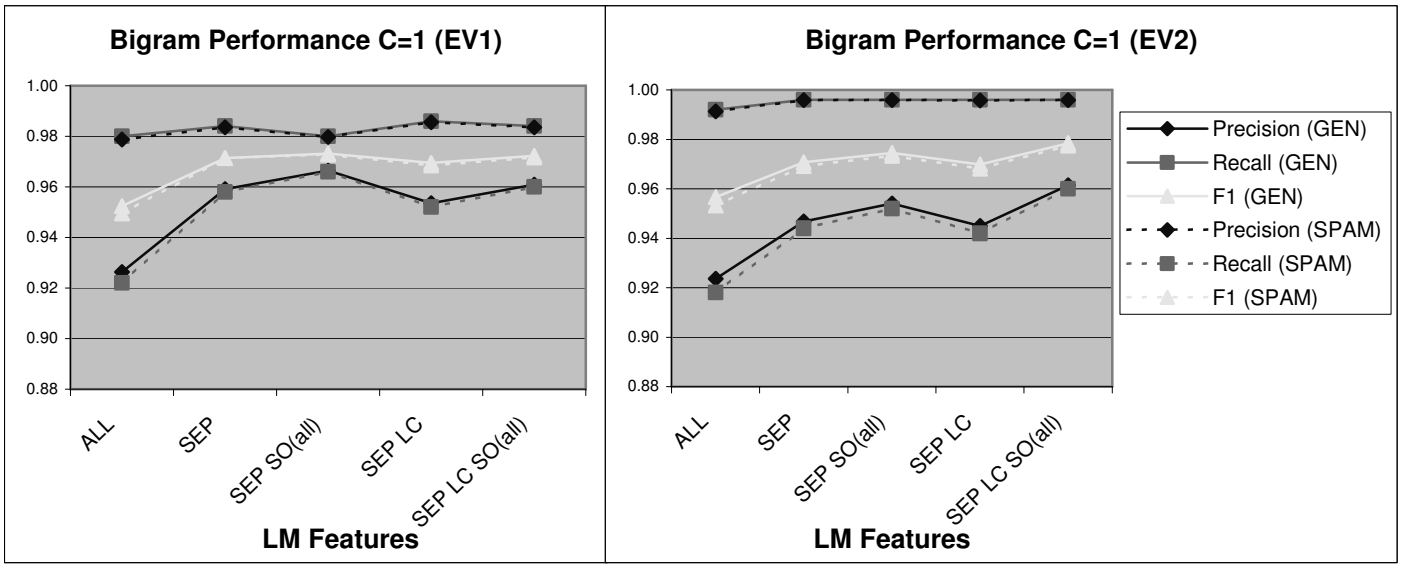

Figure 5.5: Phase 1 bigram results – EV1 and EV2

We make the following observations:

- As in the unigram case, modelling the fields separately is clearly beneficial.
- Compared to the unigram case, there is slightly more of a spread between GEN and SPAM performance (especially for EV2). This could be due to the fact that the language of genuine email tends to be more consistent than that of spam, especially when the spam is interspersed with various extra-linguistic features such as random characters and deliberately misspelled words.
- We observe similar trends to those found in the unigram results in terms of the performance benefit gained by removing single occurrences. Note that this refers to single *unigram* occurrences; the bigram cut off point (C, set at 1 for these results) controls the removal of bigram occurrences.

Converting to lower case has a slightly more positive impact in the bigram case, especially on SPAM recall (and thus GEN precision). This could be due to the extra contextual information in the bigram model making up for the lost case information. As in the unigram case, time could be spent devising methods of doing linguistically-motivated partial lower case conversion to maximise useful information retention.

Both *SEP SO(all)* and *SEP SO LC(all)* performed well in these experiments and were carried through to phase 2.

We also tested a range of values for the cut off point, C. In the following graph, we plot the performance measures for C = 1..5 using the *SEP SO LC(all)* LM feature set:

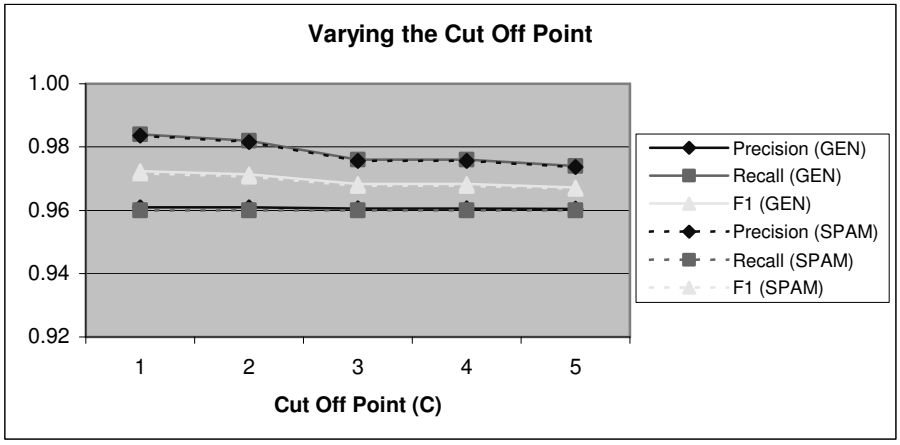

Figure 5.6: Performance of bigram models with varied cut off

Note that classification performance for GEN degrades significantly, while performance for SPAM degrades only very slightly. We suspect that this is due to the higher level of 'noise' in the SPAM data, making the lower-frequency SPAM bigrams less reliable than the GEN ones.

It turns out that as well as using C to control the absolute size of the LM (see 4.3.2), we can also use it to balance LM's with relative size discrepancies, which could otherwise lead to classification errors. For example, if the SPAM LM is significantly larger than the GEN LM (as it is in our case) then we would expect that the SPAM LM would recognise proportionally more of the SPAM language than the GEN LM would recognise of the GEN language, and thus have an 'unfair advantage' in classification terms. However, by raising the cut off point for the larger LM, we effectively reduce its coverage, balancing it more effectively with the smaller LM. In our experiments, we found that although raising the value of C for both LM's reduced performance, raising it marginally for SPAM alone increased performance. We will call this technique *LM balancing*.

Another issue that arises when optimising LM's for classification is that of estimating the probability of unknown words. In more traditional language modelling domains, such as speech recognition, the actual probability assigned to unknown words is practically irrelevant, since the probability of an unknown word across a given set of potential word sequences is constant. However, in the classification domain, the probability of an unknown word is conditional on the class being modelled. Thus, the *relative* probability of an unseen word given a class can be estimated. For example, in our case, the nature of spam is such that messages often contain many anomalous character sequences and deliberately misspelled words, whereas genuine email tends to be drawn from a more stable vocabulary. We can use this knowledge to

assign a slightly higher probability to the event of encountering an unknown word in the SPAM LM, as compared with the GEN LM. The following graphs illustrate the effect of informed class-conditional unknown word probability modelling on evaluation data performance:

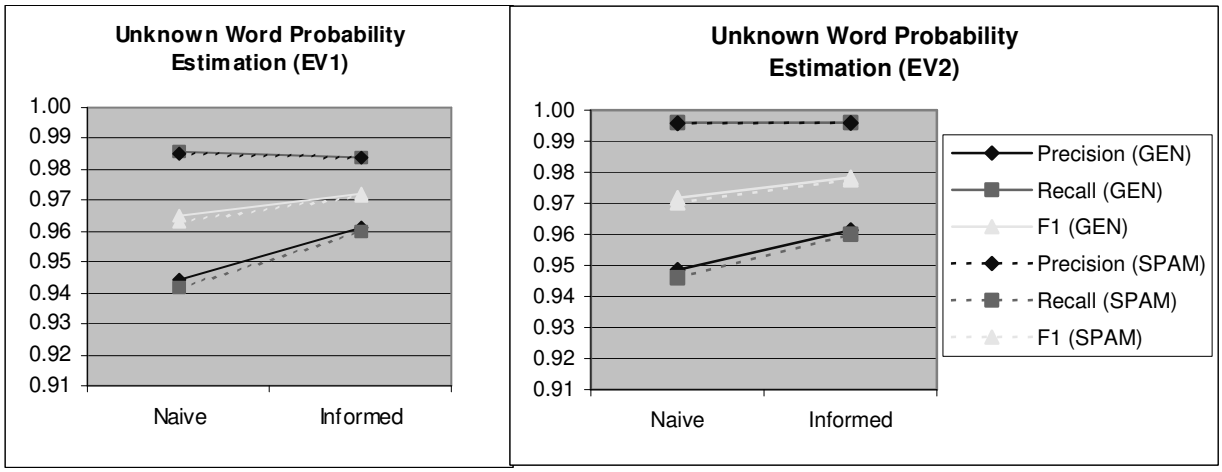

Figure 5.7: Unknown word probability estimation (SPAM floor raised from 1*e-8 to 1*e-7 for *Informed*)

By slightly raising the probability of unknown words within the SPAM LM, recall for SPAM increases significantly, while recall for GEN is affected only slightly for EV1 and hardly at all in for EV2. Interestingly, our experiments suggest that while informed unknown word probability estimation has a significant bearing on bigram performance, it does not have the same positive impact on unigram performance. We speculate that the relative probability distance between bigrams and unigrams in the bigram model (only the discounted probability mass is distributed over the unigram entries) makes unknown word estimation more suitable, while for the unigram model, adjusting the unknown word probability has too direct an impact.

Phase 2

Following the experimental method in 5.3.3 we ran the phase 2 experiments on the unseen test data set (TS). The non-adaptive results are as follows (for readability, results are reported in % and the best results in each column are emboldened):

	GEN			SPAM			Subject
LM's	Recall	Precision	F1	Recall	Precision	F1	Weight
Unigram (SO)	96.68	96.17	96.42	96.36	96.85	96.60	0.50
Bigram (GT SO)	96.55	**97.20**	96.87	**97.37**	96.76	97.06	0.65
Bigram (GT SO/LC)	96.02	97.18	96.87	**97.37**	96.28	96.82	0.70
Bigram (CN SO)	**96.95**	96.44	96.69	96.12	97.10	96.86	0.70
Bigram (CN SO/LC)	**96.95**	95.81	96.37	95.98	97.08	96.53	0.70
Bigram (CL SO)	**96.95**	96.95	**96.95**	97.11	**97.11**	**97.11**	0.70
Bigram (CL SO/LC)	96.29	96.54	96.41	96.74	96.50	96.61	0.70

Note that different backing-off techniques were tested, denoted by:

- GT = Katz smoothing with Good-Turing / linear discounting mix (GT ≤ 8)
- CN = Non-linear Confidence smoothing ($\omega = 0.1 \quad \phi = 1 - n_3/T$)
- CL = Linear Confidence smoothing ($\omega = 0.1 \quad \phi = 1 - n_3/T$)

The two optimal bigram feature sets from phase 1 (denoted by SO and SO/LC) were also included in the tests. The bigram LM's were *balanced* and informed unknown word probabilities were used (see phase 1 results).

The adaptive component was then trained and interpolated with the static component in accordance with the phase 2 experimental method (5.3.3). Optimal interpolation weights for the adaptive component cannot be estimated prior to classification of the unseen data (unless the adaptive training set is large enough, which it is not in our case); rather it must be set by empirical observation. The following graphs show the performance of the various models on the unseen test data when the adaptive component is incorporated:

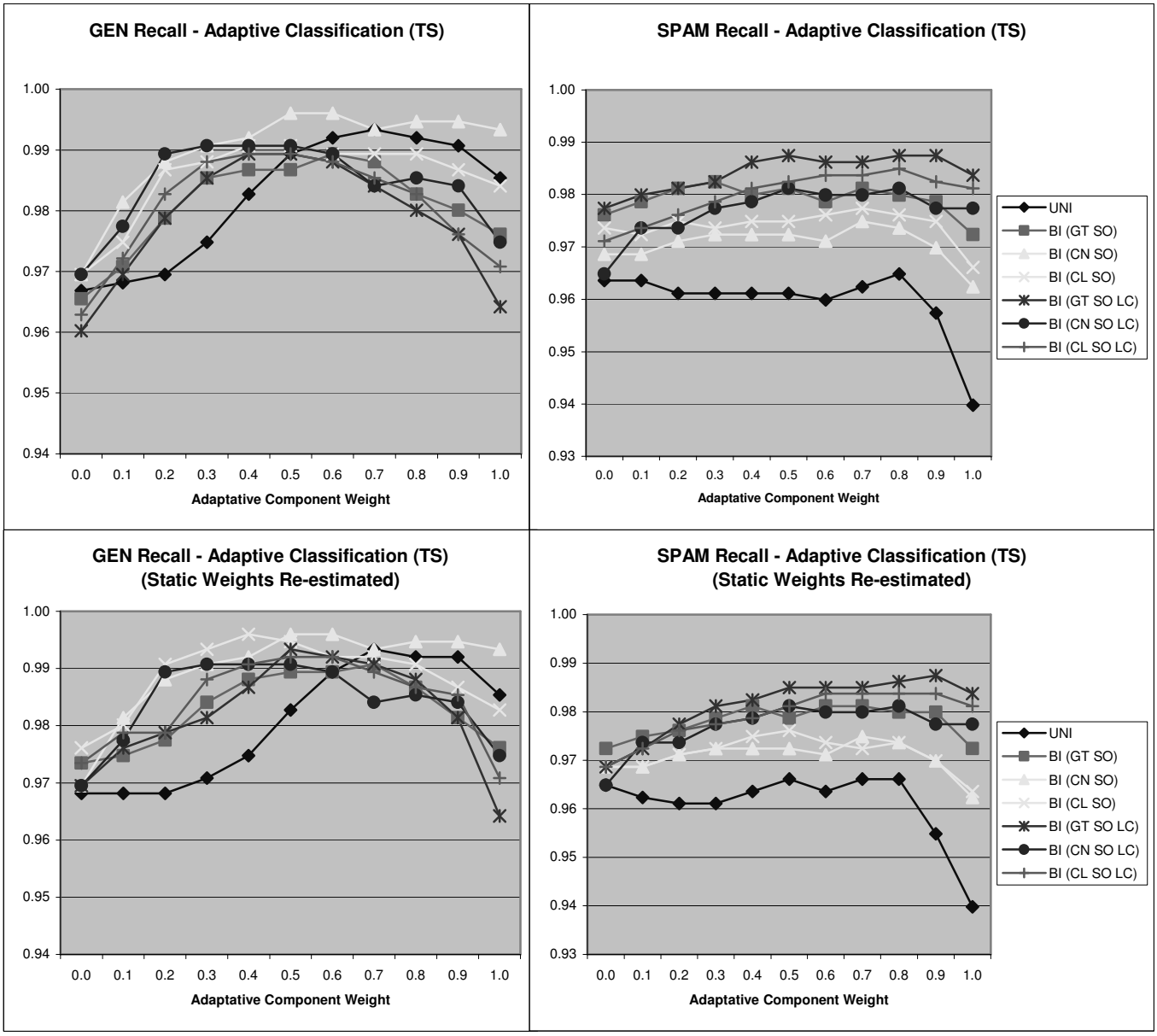

Figure 5.8: Adaptive component interpolation (Recall)

Interpolation of the static and dynamic models clearly enhances recall, while re-estimating the static component weights only results in a slight improvement for some models. It is interesting to compare the different shapes of the graphs; the GEN recall graphs are significantly more humped than their SPAM counterparts, suggesting that combining static and dynamic information about genuine email is

particularly important. We would expect this to be the case, as genuine email is likely to vary from user to user much more than spam.

The following graphs plot the F1 values for the same models:

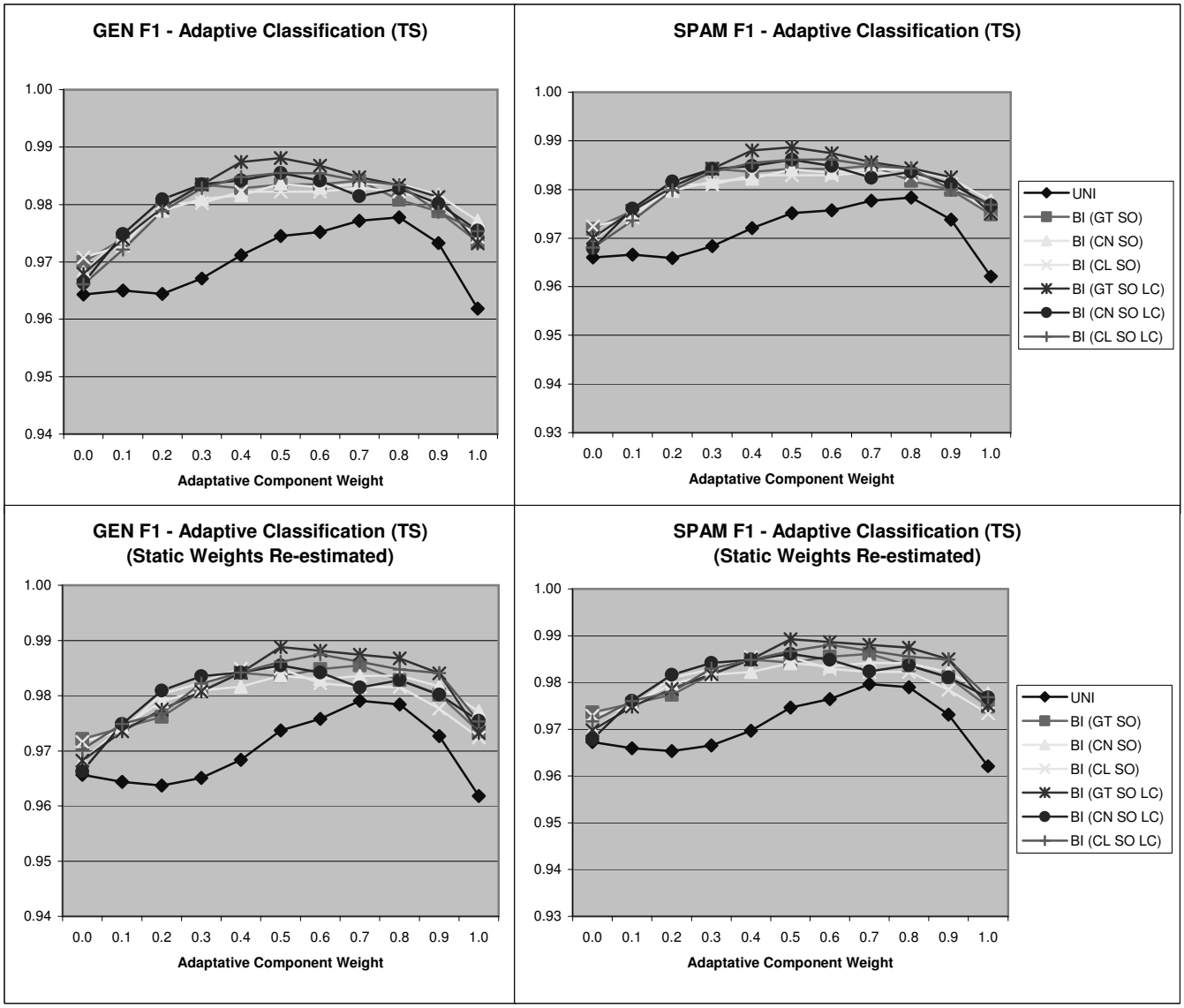

Figure 5.9: Adaptive component interpolation (F1)

We observe that the performance of the unigram models is clearly inferior to that of the various bigram models, though only by around 1-2%. Optimal interpolation weights for the static and dynamic components vary somewhat depending on the models used, though a mid-range value of around 0.5 - 0.6 seems to be optimal, as summarised in the following table:

	GEN			SPAM			Dynamic
LM's	Recall	Precision	F1	Recall	Precision	F1	Weight
Unigram (SO)	99.34	96.52	97.91	96.61	99.23	97.96	0.7
Bigram (GT SO/LC)	99.34	**98.42**	**98.88**	**98.49**	**99.37**	**98.93**	0.5
Bigram (CN SO)	**99.60**	97.03	98.30	97.11	99.61	98.35	0.5
Bigram (CL SO/LC)	99.20	98.29	98.75	98.37	99.24	98.80	0.6

For a realistic spam filtering system, an upper bound on error-rate for GEN of around 1% would probably be reasonable, requiring a classifier to obtain at least 99% recall on genuine email. If such a result is achieved, it can be assumed that in combination with a *whitelist*, the chances of the system filtering out a genuine email are very slight indeed. All of the above results meet this criteria, with a corresponding error rate for spam of around 1-3%.

Higher-Order N-grams

In addition to the results above, we also ran experiments using trigram LM's, but found a significant decrease in performance, especially for SPAM recall. We strongly suspect that given the amount of training data and the generally 'noisy' nature of email text (especially in the SPAM case), > 2-gram LM's do not generalise well enough to be effective.

Error-Analysis

Analysing the errors made by the bigram classifiers, we discovered that most of the misclassified genuine messages were either circulars or impersonal announcement messages. Conversely, the misclassified spam messages were often those written in a personal style. A combined classifier/whitelist approach would certainly limit the former, while the latter could perhaps be checked by better language modelling techniques combined with more training data.

Unanonymised Data

Due to the requisite anonymity of the *GenSpam* corpus, all of the *GenSpam* experiments were carried out using anonymised data. To ensure that performance wasn't reliant on the anonymisation process, we also ran some experiments using an unanonymised version of the *GenSpam* corpus. The results suggest that unanonymised data performs slightly better, which is what we would expect bearing in mind the extra information it contains:

	GEN			SPAM			Dynamic
LM's	Recall	Precision	F1	Recall	Precision	F1	Weight
Bigram (GT SO/LC)	99.07	**98.81**	98.94	**98.87**	99.12	98.99	0.6
Bigram (CN SO/LC)	99.34	98.17	98.75	98.24	99.37	98.80	0.6
Bigram (CL SO)	**99.47**	98.55	**99.01**	98.62	**99.49**	**99.05**	0.7

5.4.2 LingSpam Results

We present the *LingSpam* results primarily for comparison with previous work. We experimented with various LM types/features, and present the best results obtained for unigram and bigram LM's:

	GEN			SPAM		
Classifier	Recall	Precision	F1	Recall	Precision	F1
NB	–	–	–	82.35	**99.02**	89.92
kNN	–	–	–	88.60	97.40	92.79
Stacking	–	–	–	91.70	96.50	93.93
TreeBoost	–	–	–	97.92	98.33	98.12
LM (unigram)	99.33	99.58	99.46	97.92	96.71	97.31
LM (bigram GT/C=4)	99.71	99.83	99.77	99.17	98.55	98.86
LM (bigram CN/C=3)	**99.75**	99.79	99.77	98.96	98.75	98.86
LM (bigram CL/C=4)	99.70	**99.88**	**99.79**	**99.38**	98.55	**98.96**

The cut off point (C) used is indicated; this was applied only to the message body LM's; the subject LM's were constructed with the default cut off point of 1. By balancing the body LM's and raising the required classification margin in favour of genuine messages, we noted a further improvement in performance:

	GEN			SPAM		
Classifier	Recall	Precision	F1	Recall	Precision	F1
LM (bi CL/C=4,3/M=2.0)	99.79	**99.88**	**99.83**	**99.38**	98.96	**99.17**
LM (bi CN/C=4,3/M=2.0)	**99.83**	99.83	**99.83**	99.17	**99.17**	**99.17**

The bigram-based LM classification techniques we have presented clearly outperform the other techniques in these tests.

Ch 6. Discussion

In conclusion, we discuss some of the issues encountered during the project whilst attempting to look objectively at the knowledge gained and point towards further avenues of research, as well as considering how the techniques presented might be deployed in the real world.

6.1 Perplexity

Perplexity is a measure (amongst other things) of the amount of information about a given data set encoded in a given language model. The idea is drawn from information theory and is related to the concept of entropy, a measure of the *informativeness* of a distribution. There are actually two definitions of perplexity, but for our purposes, *test set perplexity* (or average log probability) is the most useful. It is defined as:

$$PP = -\frac{1}{K}\sum_{i=1}^{K}\log_2\left(\tilde{P}(w_i \mid h_i)\right)$$

where K = the number of words in the test set

$\tilde{P}(w_i \mid h_i)$ = the LM probability of word w_i given history h_i

The assumption exists that the data the LM is trained on is disjoint from the data the perplexity measure is calculated from. In theory, the lower the perplexity the more effective a language model is likely to be at carrying out the required task on the given test set. In the field of speech recognition perplexity is often used to compare the effectiveness of different LM's as an inexpensive alternative to running the recogniser and gathering error rates. In our case, calculating perplexity for a test data set is only marginally less expensive than running the classifier and calculating performance measures, so its utility from that point of view is minimal. However, we were interested in experimenting with perplexity values, and hypothesised that classification performance on a given data set should be proportional to the difference in perplexity between the two class conditional LM's on that data set, or more formally:

Given classes $C, \overline{C}$ and assuming data set of C instances:

$$recall(C) \propto PP^{\overline{C}} - PP^{C}$$

i.e. we seek to minimise the test set perplexity for class *C* and maximise it for $\overline{C}$ (note that $\overline{C}$ could be extended to arbitrarily many alternative classes). This is intuitive in the sense that we would like a class conditional LM to hold a maximal amount of information about the class it is modelling, and a minimal amount of cross-over information about other classes.

Through experimentation we found this hypothesis to be untrue in general. We suspect this is because other factors not taken into account by test-set perplexity (such as variation in document structure) significantly affect classification performance. It is possible that the definition of perplexity could be extended to give some account of features such as meta-level structure, but given its marginal utility for the classification task anyway, we deemed it an unnecessary avenue of research.

6.2 Strengths and Weaknesses of our Approach

Some perceived strengths of the classification model we have developed are:

- Training is relatively simple – complexity linear in the size of the training set (although a more advanced LM type might require more complex training algorithms).
- The classification framework enables a wide range of linguistic and non-linguistic information to be taken into account (i.e. document structure, unknown word probability etc.).
- More advanced syntactic/semantic language modelling can easily be incorporated.

And weaknesses:

- LM's can become large, potentially incurring high storage costs.
- Model-based framework can require a significant amount of parameter tuning to achieve optimal performance.
- Domain-specific parameter modelling makes the technique sensitive to structural domain changes.

6.3 Deployment

There are further issues to be faced when considering deploying technology such as that presented here to fight spam in the real world. An effective language-based anti-spam system must consist of both a front-end component to convert raw email data into the required format, and a back-end analysis engine (such as the classifier we have presented). In our case we would require the front-end to reliably extract structure and text from email messages; not an easy task, bearing in mind the ever-evolving tricks pulled by spammers to hide textual content. Ideally the front-end would also be able to extract text from a wide range of attachment formats, such as postscript, PDF, JPEG etc. For image files this would require some form of OCR (*Optical Character Recognition*) or similar. Although an advanced front-end component would be desirable, it should be borne in mind that the vast majority of email contains a reasonable proportion of plain text, and a classifier is likely to be quite effective with a fairly simple front-end.

An alternative approach is to use raw email data as input to the classifier. This has the potential advantage of implicitly taking super-textual structure into account, and pattern classification techniques for non-textual information are well-studied. However, low-level formatting discrepancies could lead to potential problems, and this approach doesn't allow for explicit, informed modelling and weighting of email-specific structure.

One advantage of the adaptive approach we have presented is in terms of deployment into settings such as online *webmail* services (often worst-affected by spam). In this scenario, the static models would reside on a central server where they would be updated infrequently and used across multiple accounts, while a set of small (i.e. ~500k total) adaptive models could be trained for each individual user and updated on a regular basis. This would maximise storage efficiency, whilst maintaining high levels of performance.

6.4 Related Further Research

Email is rich in structure, and there were a number of features that we didn't have time to build into the classifier. For example, the *To*, *From* and *Date* fields all contain information that can be used to train more accurate models. Similarly, a more accurate distribution of the MIME structure (such as number/type of attachments, etc.) of email could be constructed and used to enhance performance. Additionally, we do not take the *length* of email into account when classifying, but that can also provide clues as to whether or not it is spam.

Language modelling is a continually advancing field, and there are many ways in which current techniques could be improved upon. Research includes developing tractable methods of incorporating informed linguistic structure into LM's, both syntactic and semantic, as well as seeking to enhance the formal and statistical models that underlie the field.

Given time, we would like to carry out a more thorough analysis of the effectiveness of the discounting methods presented. Our experiments suggest that the Confidence smoothing techniques we have presented perform better with limited data, whereas Katz smoothing performs better with larger quantities of training data, though our results are by no means conclusive on this matter.

We would also be interested in assessing the effectiveness of our LM classification model in other domains, such as topic or sentiment classification.

References

Agichtein E. and Gravano L. (2001) **Snowball: Extracting relations from large plain-text collections**. In *Proceedings of the 5th ACM International Conference on Digital Libraries (DL'00)*

Androutsopoulos I., Paliouras G., Karkaletsis V., Sakkis G., Spyropoulos C. and Stamatopoulos, P. (2000a) **Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach**. In *Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000).*

Androutsopoulos I., Koutsias J., Chandrinos K. and Spyropoulos C. (2000b) **An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages**. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160--167

Carreras X. and Marquez L. (2001) **Boosting trees for anti-spam email filtering**, In *Proceedings of RANLP2001*, Tzigov Chark, Bulgaria, pp. 58-64

Chen S. F. and Goodman J. T. (1996) **An empirical study of smoothing techniques for language modeling**. In *Proceedings of the 34th Annual Meeting of the ACL.*

Church K.W. and Gale W.A. (1991). **A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams**. *Computer Speech and Language*, 5:19-54.

Drucker H., Wu D. and Vapnik V.N. (1999) **Support vector machines for spam categorization**. *IEEE Trans. On Neural Networks*, 10(5):1048-1054

Graham P. (2002) **A Plan for Spam.** http://www.paulgraham.com/spam.html

Fürnkranz J. (1998) **A Study Using n-gram features for Text Categorization**. Technical Report OEFAI-TR-98-30, Austrian Research Institute for Artificial Intelligence, Vienna, Austria.

Han S., Kumar V., Karpyris G. (1999) **Text categorization using weight adjusted k-nearest neighbour classification**. Technical report, Dept. of CS, University of Minnesota.

Hiemstra D. and de Vries A. (2000) **Relating the new language models of information retrieval to the traditional retrieval models**, *CTIT Technical Report TR-CTIT-00-09*, May 2000, ISSN 1381-3625

Huang X., Acero A., Hon H. and Reddy R. (2001) **Spoken Language Processing: A Guide to Theory, Algorithm and System Development,** Prentice Hall PTR, New Jersey 07458

Jelinek F. and Mercer R.L (1980). **Interpolated estimation of Markov source parameters from sparse data**. In *Proceedings of the Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands: North-Holland

Katz S.M. (1987) **Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer**, IEEE Trans. ASSP, 35(3)

Kessler B., Nunberg G. and Schutze H. (1997) **Automatic Detection of Text Genre**. In *Proceedings of 35th Annual Meeting of Association for Computational Linguistics and 8th Conference of European Chapter of Association for Computational Linguistics*, pages 32--38, Madrid, Spain.

Lewis D. (1992) **An evaluation of phrasal and clustered representations on a text categorization task**. In Croft et. al. (Ed.), *Proceedings of SIGIR-95, 15th ACM International Conference on Research and Development in Information Retrieval* (pp. 37-50). New York: ACM Press

Lewis D. and Ringuette M., (1994) **Comparison of two learning algorithms for text categorization**. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval (SDAIR'94)*

McCallum A. and Nigam K. (1998) **A Comparison of Event Models for Naive Bayes Text Classifcation**. In *Proceedings of AAAI-98 Workshop on "Learning for Text Categorization"*, AAAI Press

Mikheev A., Grover C. and Moen M. (1998). **Description of the LTG System Used for MUC-7**. In *Proceedings of the MUC-7*

Mikheev A. (1999) **A knowledge-free method for capitalized word disambiguation**. In *Proceedings of the 37th Conference of the Association for Computational Linguistics (ACL'99)*, pages 159-168. University of Maryland.

Mikheev A (2002) **Periods, Capitalized Words, etc.** *Computational Linguistics* 28(3): 289-318

Ng H.T., Goh W.B. and Low K.L. (1997) **Feature selection, perceptron learning, and a usability case study for text categorization**. In *20th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'97)*, pages 67--73.

Ng A.Y. and Jordan M.I. (2002) **On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes.** In T. Dietterich, S. Becker and Z. Ghahramani (Eds.), *Advances in Neural Information Processing Systems* (NIPS) 14.

Pang B., Lee L. and Vaithyanathan S. (2002) **Thumbs up? Sentiment Classification using Machine Learning Techniques**. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*

Ponte J.M., Croft W.B., (1998) **A language modelling approach to information retrieval**. In *Proceedings of the 21st ACM SIGIR Conference on Research and Development in Information Retrieval*.

Rennie J. (2000) **ifile: An Application of Machine Learning to E-Mail Filtering**. In *Proceedings of the KDD-2000 Workshop on Text Mining*

Runbinstein Y.D. and Hastie T. (1997) **Discriminative versus informative learning**. In *Proceedings of the Third Int. Conf. On Knowledge Discovery and Data Mining*, pp. 49-53. AAAI press.

Sahami M., Dumais S., Heckerman D. and Horvitz E. (1998) **A Bayesian Approach to Filtering Junk E-Mail**. *Learning for Text Categorization* – Papers from the AAAI Workshop, pages 55–62, Madison Wisconsin. AAAI Technical Report WS-98-05.

Tan C.M., Wang Y.F. and Lee C.D. (2002) **The Use of Bigrams to Enhance Text Categorization Performance**. *Journal Information Processing & Management.* , vol. 30, No. 4, pp. 529-546

Vapnik V.N. (1998) *Statistical Learning Theory*. John Wiley & Sons, 1998

Wiener E., Pedersen J.O., and Weigend A.S, (1995) **A neural network approach to topic spotting**. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95)*

Yang Y. and Chute C.G. (1994) **An example-based mapping method for text categorization and retrieval**. *ACM Transaction on Information Systems (TOIS)*, 12(3):252-277

Yang Y. and Liu X. (1998) **A re-examination of text categorization methods**. School of Computer Science, Carnegie Mellon University

Yang Y., and Pedersen O. (1997) **A Comparitive Study on Feature Selection in Text Categorization**. *Proc. of the 14th International Conference on Machine Learning* ICML97, pp. 412 – 420.

Yerazunis W. (2003) **Sparse Binary Polynomial Hashing and the CRM114 Discriminator**, *Mitsubishi Electric Research Laboratories Cambridge*, MA 02139 USA

Appendix A – Replaceable PoS Tags (CLAWS2):

NP1
NP2
NN
NNL
NNS
NNT
NNU
JJ
RR
VV*

Appendix B – Implementation Code:

- Our classifier implementation is written in C++, and consists of the following header and class files:

Classify.cc
CalcDocScore.cc
CalcDocScore.hh
EmailReader.cc
EmailReader.hh
LMUtils.hh
LM.cc
LM.hh
Unigram.cc
Unigram.hh
Bigram.cc
Bigram.hh
Trigram.cc
Trigram.hh

They are located in the following directory: **/homes/bwm23/project/code/classifier**

- Our LM construction tools are written in perl, and include the following:

make_unigram
make_bigram
make_trigram

They are located in the following directory: **/homes/bwm23/project/code/LM**

- Our anonymisation procedure makes use of the following perl scripts:

generate_patterns.pl
anonymise_patterns.pl
anonymise_man_patterns.pl
prop_anons.pl

These, and other perl and shell scripts used in the formatting and anonymisation process are located in the following directory: **/homes/bwm23/project/code/scripts**