Improving Query Term Expansion With Machine Learning

Vaughn Wood

a thesis submitted for the degree of

Master of Science

at the University of Otago, Dunedin,

New Zealand.

Contents

List of Tables

List of Figures

Abstract

Vocabulary mismatch is an impediment to responding to user queries with relevant results. Stemmers solve this problem by conflating terms with similar spellings. In this thesis we use machine learning to create a stemmer optimised for Information Retrieval performance. We investigate further improvement to stemmers with corpus information. With the goal of stemming selectively for further performance gains we investigate the prediction of query performance.

Chapter 1

Introduction

Search engines aim to help users find information quickly. With a given query that describes an information need, the search engine provides a ranked list of relevant items with respect to that query. With ad-hoc information retrieval the ranked items are documents and the query is short and in human language.

A typical search engine finds those documents that contain the query terms. To rank the documents the search engine reduces each document to a score. A document's score should indicate how relevant it is. Reducing each document to a score allows us to look at each document without examining others. Some words occur in almost all documents and fail to provide information about the contents of documents so we discount their importance.

Queries are as ambiguous and incomplete as the words they are made from. Due to homographs, synonyms, words with multiple meanings, and multiple forms of a word, any word could describe more than one information need, or only part of one. For example, the query 'bear fruit' would produce documents about the eating habits of bears and documents about successful plans. The search engine may find irrelevant documents since terms in queries and documents are only tokens. As tokens, words that share the same spelling are indistinguishable. Words that describe the same concept, but are spelt differently, are ignored.

Relevant documents may evade the engine because they only contain different words to describe the same concepts. Even when a document does contain a query term, it might not score as high as it should because some words that indicate the right topic are not in the query. This is vocabulary mismatch. The vocabulary used by the query does not match that used by the document. For example, searching for 'meat' will not find a document that only uses 'beef' and 'mutton'.

1.1 Stemming

One way of preventing vocabulary mismatch is to treat word forms that describe the same concept as being the same word. Those word forms are conflated by the search engine. Different approaches to finding these word forms include using a thesaurus, removing affixes, and using collection statistics to form classes of equivalent words.

Spärck Jones (1971) discovered groups of words that describe the same concept by examining the co-occurrence of the words in the collection. Each document containing two words together provides evidence that those words share a meaning. Every time one occurs and another fails to occur provides evidence for separate meanings.

As much as these approaches can help some queries, other queries can be hurt by them. Query drift occurs when query terms are conflated with terms irrelevant to the query. Documents related to the irrelevant terms will occur in the list of documents returned by the search engine. This hurts the performance of the search engine.

Stemming is the transformation of words into their stem. The stem is the main part of a word without suffixes. For information retrieval we try to remove suffixes that barely alter the meaning of the word. Plurals are a good example. The suffix '-s' pluralises words, but to a user little has changed.

Stemmers have been improved in the past by ensuring that potentially conflated terms occur together often in documents (Xu and Croft, 1998). This can prevent common mistakes that stemmers make. For instance, one mistake made by the stemmer discussed in chapter 3 is to conflate 'opera' with 'operand.' This mistake could be avoided as these words only occur together once in the collection, much less than chance suggests. This method refines stemming by using information about the language used in the collection. Collection information is not normally used by stemmers, which just base decisions on the letters in the word to be stemmed.

1.2 The State of Stemming

All stemmers are based on written language. Soundex (Jacobs, 1982) and Metaphone (Philips, 1990) work similarly, but attempt to conflate words with similar nnsounds rather than meanings. The Lovins Stemmer (Lovins, 1968) was created by hand in 1968. This was the first rule based stemmer and influenced the others.

Lovins created the stemmer based upon engineering vocabulary. This shows that stemmers can be specialised. This is of importance to us as being able to learn stemmers means we can tailor them to different English collections. Lovins' stemmer consists of a large series of suffixes to replace with others. Only the longest matching rule is used, and not all rules remove suffixes. Some of the rules replace one suffix with another, or prevent a shorter suffix from being removed.

The Porter Stemmer (Porter, 1980) is still frequently used in Information Retrieval (IR). It provides multiple passes of rules that replace smaller suffixes. Instead of removing 'ings' Porter would remove 's' and then 'ing'. These rules are concise, and flexible. Porter uses complex conditions upon these rules to avoid making mistakes. These rules can be seen in Appendix A.

Porter created Snowball (Porter, 2010), a language to describe stemmers. It gives a way of describing stemmers so they would not need to be implemented repeatedly.

The reason for those rules was that they made sense, not that they directly improved search engine performance. Stemmers can focus on correctness of the words, or on search engine performance. It is difficult to create rules that improve search engine performance by hand. The result of searching large document collections is hard to predict, especially without knowing what queries may be made.

There are other approaches in this area. One is to use the co-occurrence of terms to determine which terms should be conflated. We elaborate on this in Chapter 4. This has one difficulty that rules do not, algorithms may generalise better where there is little information. Co-occurrence tends to fail with rare words, where there is less information. Rules that transform words do not.

Thesauruses are also used to determine which terms have similar meanings and may be conflated. This does not deal well with expanding vocabularies and newly coined terms. There is no guarantee that this method is effective for Information Retrieval performance.

Spärck Jones (1971) used clustering terms to discover terms to add to queries. Expanding queries is similar to, but not identical to, stemming. Conflating terms rather than adding terms to a query has a different effect on retrieval. Ranking measures score them differently. Using term co-occurrence is similar to clustering terms.

A common statistical approach is to use n-grams (Mayfield and McNamee, 2003). n-grams are contiguous blocks of text. In the case of stemming, grams are blocks of letters. Given a small number of letters, each block in a word is examined, and

the frequency in the vocabulary is determined. An n-gram stemmer expects root words to be rarer than prefixes and suffixes. Blocks that occur with a much greater frequency are removed. This approach is language-neutral.

1.3 Predicting Performance

Query Performance Prediction (QPP) is an attempt to predict how queries perform. If we can predict how stemming will affect performance, we can apply it selectively. So predicting query performance can further refine the effects of stemming upon search engine performance, if the predictions allow us to stem queries that will be improved. Previous work (Cronen-Townsend, Zhou, and Croft, 2002a)(Macdonald, He, and Ounis, 2005)(Grivolla, Jourlin, and Mori, 2005)(Hauff and Azzopardi, 2009) has focused on how well the predictors correlate with measures of performance, and how they might be used for selecting when to expand queries.

Performance of a search engine in answering a query can be quantified if we know the relevant documents in a collection. QPP is doing well if the predicted performance is similar to measured performance. We settle for correlation between predicted and measured performance as predictions are useful as long as they can distinguish queries that will perform well from ones that will perform badly. If we can predict how well queries perform with and without stemming then we can choose the best method for each query.

1.4 Machine Learning in IR

Our approach to solving the problems of vocabulary mismatch, query drift and when to stem is to use machine learning. Information Retrieval has a history of Machine Learning (Trotman, 2004). Once we have a way to quantify how well queries perform, we can approach these problems as optimisation. Measures have been created to do so.

Machine learning is often used to optimise aspects of the retrieval process. Genetic Algorithms (GA) and Genetic Programming (GP) have been used (Gordon, 1988)(Trotman, 2005). Both are forms of parallel search which work on representations of solutions which can be compared for relative fitness. We optimise the search method for information retrieval performance. Performance is measured using previously judged queries (Trotman, 2004).

Focusing on the techniques used in this thesis, Genetic Algorithms have been used by Gordon (Gordon, 1988) to learn document descriptions. He treated the descriptions as bags of bitstrings, each bit indicating which keywords to apply to a document. Genetic Programming has been used to learn ranking functions such as those learnt by Trotman (2005). Learning of mathematical functions is a classic application of Genetic Programming (Koza, 1992).

1.5 Goals

The goal of this thesis is to use machine learning to improve the state of stemming. Particularly the performance impact of stemming on search engines. We approached this as a series of learning tasks. First we learn a stemmer using a genetic algorithm. Then we learn to improve it using term similarity measures and query performance prediction. By doing so we aim to reduce vocabulary mismatch, find more documents, and improve ranking. Using symbolic methods of learning gives us insights into the language of documents. A further goal is to create a process to automatically learn rule-based stemmers.

The questions we seek to answer are "Can we learn a stemmer focused on performance using rated queries?", "How well would such a stemmer perform compared to existing stemmers?" and "Can we further improve performance with information from the collection being searched? or from the retrieval process?"

Our method for learning a stemmer was chosen to be appropriate for other collections and other languages with properties similar to English. These would be languages like Spanish which share inflectional morphology; languages where affixes alter a word stem. Our methods for improving stemmers should work with other stemmers. Predicting stemmer performance may be tied to the stemmer used, but a method that gives significant improvement should also give insight into this kind of prediction.

Chapter 2

Machine Learning Experiments in Information Retrieval

This chapter describes the concepts used in later chapters. First we outline the method of information retrieval experiments using a corpus and queries with rated documents. This method allows us to measure performance, for both feedback in our machine learning and reporting results. We can treat stemmers as different methods of search and compare performance in a similar manner. Then we introduce how stemmers function and their properties.

We then look at the methods used to learn stemming, and improve stemmer performance. Genetic Algorithms will be used to learn a stemmer. We use patterns of word occurances to determine how similar terms in the collection are, and as a further refinement to stemming. Finally, we hope to predict stemming performance on queries sufficiently to be able to choose when to stem and thereby improve IR performance.

2.1 The Cranfield Paradigm

The Cranfield paradigm (Voorhees, 2002) is a common way of performing information retrieval experiments. We choose a corpus of documents. We create queries that include both terms for the search engine and a description of the information needed for the query to be fulfilled. Human judges use those descriptions to determine if documents are relevant. Knowing the relevance of documents we can compare and quantify the performance of search methods.

If we had an automatic method of judging documents for relevance, search would

be solved. Sadly, assessing documents for relevance is a human-intensive task, and assessing every document for relevance for every query is a large task. The INEX 2009 Wikipedia collection consists of more than 2 million documents. Judging even a few thousand documents for a single query can take hours.[1]

We can restrict assessing to those documents that actually appear in results of participant search methods for the query. Since good performance measures weight the lists towards the top, the top n documents of each list make up the pool of documents to be evaluated by the participants. So the participants must submit results for all the queries before any assessing can happen. Then the assessments allow comparison between the search techniques used.

Since we need the assessments, we use the collection after experiments have taken place. Afterwards the assessments are available to measure performance for machine learning. Sets of queries and assessments from other experiments are used to validate and test the results of the learning.

There is a problem with using judgments of only part of the collection for learning stemming. Stemming aims to discover further documents that would not normally be found. These are the very same documents that are unlikely to be judged from the pooling process.

The large number of documents not found by the original search methods lack any information about relevance. Regarding any query, a document may be relevant, irrelevant or of unknown status. Stemmers aim to find additional documents that contain alternate forms of query terms. So some of the additional documents found are likely to be of unknown relevance. Unknown relevance is treated as being equivalent to irrelevant documents by traditional measures. This should make learning stemmers that out-perform the original search methods difficult. However stemming also affects the ranking and can increase precision as well as recall.

To ensure that our results generalise to other sets of queries, we partition the queries into three sets. The first set is used by the learning algorithm as training data. Our learning methods use this set to find stemmers, parameters to improve stemming, and predictors of stemming performance. The second set is for validation, which we use to select a final solution from the solutions found. The best is chosen based entirely on performance on the validation set, as this is a more accurate predictor of future performance than anything from the training set. Relying on the training set for choosing a solution leads to overfitting on the training set.

[1] Sadly, this is from experience.

Overfitting leads to results which work very well on the training set, but badly on other data. The final set is used to measure the performance of the result. This ensures that the learning has found genuine rules for stemming, and not rules that only improve performance for the training set.

Some later experiments have no use for a validation set, as there was only one solution. In this case we simply skip the validation set, and keep the training set and test set the same.

2.1.1 Measuring Search Engine Performance

A good search engine responds to a query with a ranked list of documents that contains relevant documents at the top. With assessments it is possible to rate how well a ranked list performs in satisfying the information need of a query. Rating performance is based on two complementary properties of a ranked list, precision and recall. These measure both sides of the trade-off in returning all the relevant documents (Perfect Recall), and returning only relevant documents (Prefect Precision).

Precision

Irrelevant documents in the results make work for the user who must search the results for relevant documents. Precision is the proportion of the found documents that are relevant to the query. It is zero when no documents are retrieved. This examines how well the topic has been answered without noise in the form of irrelevant documents. The worse precision is, the more work there is left for the user in searching the results. This depends on the number of irrelevant documents that have been retrieved.

$$Precision = \frac{|\text{retrieved and relevant}|}{|\text{retrieved}|}$$

Recall

Recall is the proportion of relevant documents that have been found. This is one in the case where all the documents are returned, and zero if none have been found. This shows how exhaustively the results list satisfies the information need. Low recall tells us a search engine is missing relevant documents. Recall tells us nothing about non-relevant documents in the results. Stemmers can improve recall by

adding relevant documents to the results list. A stemmer can never decrease recall as they cannot decrease the number of returned documents.

$$Recall = \frac{|\text{retrieved and relevant}|}{|\text{relevant}|}$$

Ranking

Since search engines are most useful when they save time, the results of a search engine are ranked. The top of the list consists of the documents that the search engine believes to be most relevant to a query. Recall and Precision ignore the ranking of results, and treat the returned documents as a set rather than a ranked list. We would like to have some way of rewarding search methods that place relevant documents higher in the results.

In order to do so, the research search engine we use, Ant, scores documents according to Okapi BM25 (Robertson, Walker, Jones, Hancock-Beaulieu, and Gatford, 1996). This is a ranking function that scores found relevant documents according to their placement in the results list.

Let Q be a query, q_i be a query term from Q, D be the document being scored, $tf(q_i, D)$ be the frequency of the term q_i in document D, $df(q_i)$ be the document frequency of the term q_i, and $avgdl$ be the average document length. Let k_1 and b be parameters that we need to find for the particular collection, $|C|$ be the number of documents in the collection, and $|D|$ be the number of terms in the document.

$$IDF(q_i) = log\frac{|C| - df(q_i) + 0.5}{df(q_i) + 0.5}$$

$$BM25_{D,Q} = \sum_{i=1}^{n} IDF(q_i)\frac{f(q_i, D)(k_i + 1)}{(f(q_i, D) + k_1)(1 - b + b\frac{|D|}{avgdl})}$$

Precision at n

Precision at n ($P@n$) is more suited than precision and recall for dealing with the ranked lists that search engines produce. It is the precision of a list that has been cut off at a particular n in the ranking. This accounts for precision in the documents that the user is most likely to look at. n is usually around ten or so, which is similar to the number of items on the first page of a web search.

Let i be the rank in the results list of a document (1 = best), and n be the length of the trimmed list of results.

$$P@n(n) = \frac{count(n)}{n}$$

$$count(n) = \sum_{i=1}^{n} isrel(i)$$

$$isrel(i) = \begin{cases} 1 & : i \text{ is relevant} \\ 0 & : \text{otherwise} \end{cases}$$

Average Precision

While $P@n$ gives a measure of what is important, it has a very sharp cut-off between documents in the top n documents and the rest. A relevant document at $n + 1$ is not rewarded at all. This gives no information to the learning algorithm that those relevant documents should be higher in the results. Much better is a score based on the rank at which relevant documents appear. This will provide better feedback about where improvements can be made in ranking. Average precision (AP) provides us with better information.

A perfect AP of one amounts to returning all the relevant documents in any order. Adding non-relevant documents to the end of the list has no effect at all. Placing non-relevant documents before the relevant ones decreases the score. Removing relevant documents decreases the score even further, until there are none and AP is zero. This is equivalent to the average of $P@n$ over all the ranks of relevant documents.

Let R be the ranked list of returned documents.

$$AP = \frac{1}{|R|} \sum_{i}^{R} P@n(i) isrel(i)$$

Mean Average Precision

To reduce the variance of our results, and to provide better evidence for differences between search methods we use the Mean Average Precision (MAP) of a set of queries. This is widely used to measure retrieval performance so we use it in our machine learning, and to report results.

MAP is the mean of all the average precisions of a set of queries. Since AP assumes unassessed documents are irrelevant, so does MAP. Let Q be the set of queries.

$$MAP(Q) = \frac{1}{|Q|} \sum_{q \in Q} AP(q)$$

Rank Effectiveness

When reusing the assessments created by previous experiments in the Cranfield paradigm documents without assessments are likely to be found. Rank Effectiveness (Ahlgren and Grönqvist, 2008) is an attempt at coping with missing relevance judgments. Where MAP assumes that unassessed documents are irrelevant, Rank Effectiveness allows them to have no impact on the scores at all. Again, this is especially important with stemming, as our goal is to find documents not found by other methods.

Let R be the ranked list of returned relevant documents, $nr(doc)$ be a function that returns the known non-relevant documents before doc in the ranked list, and NR be the set of known non-relevant documents.

$$\textit{RankEff} = \frac{1}{R} \sum_{doc \in R} \frac{|nr(doc)|}{|NR|}$$

Proportion of Queries improved

Even a new search method that improves MAP on a set of queries may not help with every query. So we measure the proportion of queries that are improved. This would normally be hidden in the MAP value, as a large improvement to a single query is equivalent to small improvements to many queries. So we examine the proportion of queries improved separately. It is the chance a search method would pay off for the user.

2.1.2 Significance test

Finding a positive difference is not enough. We want to show that our measured changes to retrieval represent a positive, systemic change to the search. Slight positive changes can be a result of noise and should not lead us to believe we have improved searching. So we perform the following significance tests on our results to confirm them.

t-Test

This measures the likelihood that a distribution could produce two samples with means at least as different as the ones we find. When it is unlikely that results from a new method could have come from the distribution of normal search we consider the new results statistically significant. Since we may change queries negatively as well as positively, when comparing query by query, the t-test should be two-tailed. As the queries remain the same for each search method the tests are paired. Smucker, Allan, and Carterette (2007) suggest it as preferable or comparable to other options, such as Fisher's permutation test and the bootstrap test, for comparing IR scores.

Bonferoni Correction for multiple t-tests

A single t-test uses a p-value to determine when to consider results significant. For us, this is when the probability of the results being from the same process is less than 5%. This chance of error is compounded if we perform multiple t-tests. The Bonferoni correction is a simple change to the significance level. We take the new significance level as $\frac{0.05}{n}$ where n is the number of t-tests performed.

2.1.3 Validation

Reporting our results on the same data that was used to train a model would give no information about how the model performs on unseen queries. As such we use a separate test set that the learning has not been exposed to for reporting final results of all models. In cases where there is a need to select between models, such as those learned during separate runs of machine learning, we use a third set to validate the models. The validation set allows us to pick a model based on their generalised performance. When we used a validation set, we always selected the model with the highest performance on validation.

2.2 Stemming

Stemmers aid search engines by increasing recall. Recall is increased by the discovery of documents the search engine would have missed without stemming. The conflation of terms by stemmers increases the number of documents a search engine

can find given a term. This set can never decrease through the use of stemming; at a minimum we always return the set of documents containing the term itself.

2.2.1 IR performance

Hull (1996) investigated evaluating stemmers and found a need for statistical tests. He also found it was difficult to show significant differences between stemmers with the typical amount of queries used in the literature. While showing an improvement over a baseline without stemming was easier. Hull also stresses the importance of looking at the results of individual queries.

2.2.2 S Stripping

A common solution for stemming in English is to remove the last 's' from every word. This usually provides conflation between words and their plurals. It also avoids most errors in stemming, by being cautious. The version used in this work has some other related rules that fix mistakes that removing the last s creates. This version is shown in Table 2.1. Only the first matching rule is used.

Table 2.1: S Stemmer

'-ies'	$\rightarrow$	'y'
'-es'	$\rightarrow$	''
'-s'	$\rightarrow$	''

2.2.3 Porter's algorithm

Porter's algorithm is a commonly used stemming algorithm that dates from 1980. It is very similar to the earlier Lovins Stemmer. Both consist of groups of substitution rules. These rules provide suffixes to match and suffixes to replace matched suffixes with. These stemmers provided the inspiration for using symbolic learning to create a stemmer.

The rules in Porter have three parts. There are two suffixes. If the first suffix matches the suffix of a word then it is replaced with the second suffix. The second suffix can be blank in order to remove the matched suffix. The third part is a condition that must be true before the rule can make any changes. For most rules in Porter

this is a check on the word's length but other checks are possible. The rules of Porter are given in Appendix A. As an example the two rules 'SS $\rightarrow$ SS' and 'S $\rightarrow$ ' from the first group are matched against query terms. If the first rule matches, it replaces the '-ss' suffix with itself. This does nothing except prevent the next rule from removing the last 's'. This allows such words as 'process' to retain their ending, while 'others' would lose the 's'.

Porter conflates words by altering the suffixes of words such that the words now match other words. The terms that Porter creates are not necessarily actual words, but can be used internally to represent classes of words to treat as equivalent.

2.2.4 Strength

Stemmers are strong if they conflate many terms together. Likewise they are weak if they conflate few terms. Porter is a stronger stemmer than the S stemmer because it conflates more terms than the S stemmer. This is obvious since the Porter stemmer contains the rules of the S stemmer.

2.3 Genetic Algorithms and Genetic Programming

Genetic Algorithms (Holland, 1975)(Goldberg, 1989) and Genetic Programming (Koza, 1992) are parallel searches inspired by evolution. Both methods are symbolic machine learning techniques that store information about the search space as a population of possible solutions. Since they are symbolic forms of learning, they are suited to analysis. A Genetic Algorithm is particularly suited for learning a rule-based stemming algorithm. The basic algorithm is given in Table 2.2.

There were several reasons for using Genetic Algorithms to learn Stemmers. A Genetic Algorithm can represent rules easily, and will give an understandable stemmer as a result. Rules are preferred to a dictionary or statistics due to their ability to work on unseen examples. Using a Genetic Algorithm also means the fitness function is directly tied to search performance, being the MAP of the queries.

Genetic Algorithms (GAs) and Genetic Programming (GP) start with a population of random solutions. How to generate an solution randomly depends on the problem being solved, and which of GAs or GP are being used. Individuals are evaluated according to an arbitrary function specified by the user. Performance on this function is used to decide which individuals to use to produce new individuals.

Select parameters
Generate an initial population
Evaluate individuals with the fitness function
Until terminating condition:
    For each individual **in** new population:
        Pick a genetic operator
        Pick parent(s)
        individual ← operator on parent(s)
    Evaluate individuals with the fitness function
Return the individual with the highest fitness

Table 2.2: Summary of Genetic Algorithms, and Genetic Programming

2.3.1 Representation

Before solutions can be learnt, we need to choose a way to represent them. This determines how learning can manipulate solutions. With the Genetic Algorithm, this has traditionally taken the form of a fixed-length bit string. Variable length strings are more complex, but still possible. The representation for Genetic Programming is an abstract syntax tree. This means the choice of representation for Genetic Programming just involves deciding upon functions and terminals to use in the tree.

We use a variable length string for representing stemmers, and an abstract syntax tree for representing formulas.

2.3.2 Genetic Operators

The algorithms share two similar operators to generate individuals from others. These are inspired by evolution. Mutation, the first operator, changes a single individual slightly. For Genetic Algorithms this is usually flipping a random bit in the bitstring. For Genetic Programming it involves replacing a single node in the tree with a newly generated node. The second operator is crossover. This mixes two parent individuals into a child individual. For a Genetic Algorithm this means taking a random point in both bitstrings. The child is created from the first part of the first parent and the second part of the other parent. For Genetic Programming it chooses a point within both parent trees, and replaces a subtree of the first parent at that point with the subtree from the second parent. Other forms of crossover are

popular, but we do not examine them.

Mutation is a way of providing diversity within the population. It provides new structure in the population. Crossover mixes structures that are already in the population. This attempts to share the fit aspects of the two parents to create fitter children. Used together they allow for exploration of the search space and for exploitation of features found in the space. Reproduction is a third operator that simply replicates an individual from the current population to the next. This allows selected individuals to remain in the population untouched.

2.3.3 Selection

We use the scores attained by the fitness measure to bias selection of fitter individuals for generating the next population. If we did not select fitter individuals more often, this would amount to a random walk of the search space.

The are two main kinds of selection. The more common kind is fitness-proportional selection. This selects individuals with a probability proportional to the amount of fitness they have compared to the total fitness in the population. This requires normalisation of the fitness values. If fitness values are not normalised, then the bias towards fitter individuals decreases as population fitness increases. Alternatively, tournament selection provides a constant amount of bias towards fitter individuals. It treats them as a ranked list, and selects the fittest individual from a tournament of random individuals. Tournament selection is used for all the following experiments.

To prevent the operators from ever losing the current best solution, we keep the current best whenever we create a new population. This is commonly referred to as elitism.

2.4 Term Similarity

Information in a collection exists about the similarity of terms. There are several methods that are commonly used, but all derive from the same source. How often terms occur together is used as an indication of how similar their meanings are. The more two terms occur in documents together, the closer their meanings are determined to be. In this thesis we use term co-occurrence within documents to determine the strength of a semantic connection between a pair of terms.

This information is valuable for stemming. When conflating two terms with a

rule-based stemmer only the characters of a term are examined. Using term similarity measures will add semantics to the syntactic information already used.

2.5 QPP for stemming

Query Performance Prediction seeks to predict how well a search method will answer a query. QPP methods are automatic and provide predictions without relevance judgments, or human help. With a good QPP method we would be able to choose the better of two possible results for a query.

One motivation for research in predicting Query Performance is to perform selective stemming. We want to limit stemmming to when it would be effective, and avoid it when it would be harmful. So far, the assumption has been made that we can use general QPP methods. Grivolla *et al.* (2005) decided to stem those queries predicted to have an already high Average Precision.

We instead want to correlate QPP measures with the improvement found using stemmers, found by using the change in AP when stemming is used from the baseline. We believe what makes a good predictor of search performance is not necessarily what makes a good predictor of when to stem.

In checking forms of QPP we note that work has been done on using them to selectively expand queries. Hauff and Azzopardi (2009) note that high correlations (greater than 0.5 using Kendall's τ) are needed which simply are not found in current measures[2]. They assume that high AP should correlate well with good performance with an expanded query. This correlation is attributed to query drift when AP is low. Query drift is the loss of specificity in the query. The query no longer has the ability to pick out relevant documents, so stemming should not help. Rather than assume this, it would be better to find measures which correlate well with good performance (with stemmers in this case).

In fact, we found the performance of queries without expansion and queries with expansion to negatively correlate (Pearson's correlation of -0.2562). This correlation is not high, but seems important. One method being good provides slight evidence that the other method will be bad. This shows deciding which method to use is important. This is probably due to the limits on Average Precision. At 0 AP we can't do any worse, and at 1 there is no possibility of improvement.

[2]Though of course an increase in performance from the query expansion should help this as well as increased correlation.

Some of these measures are preretrieval, as such they are more efficient, but have less information to work with. Using the highest Okapi BM25 score from the ranked list is a good predictor of Average Precision. Not all ranking functions have been found to correlate with Average Precision (Grivolla *et al.*, 2005). The difference between the two top scores is a good predictor of the improvement when stemming. The benefits of this should be obvious, since we would be ranking anyway we get a good measure for free without extra work.[3]

2.5.1 Kendall's τ

Kendall's τ is one of the correlation measures used by the literature for reporting QPP performance. This one is ideal as other correlations do not deal with the type of data found in the QPP scores. Pearson's correlation cannot be used due to the lack of normal distributions in the data. Spearman's rho is discounted as it is unsuitable for determining the magnitude of correlation.

We also do not mind a nonlinear correlation as long as the ranking is similar enough to allow classification. Reasonable machine learning techniques exist that can handle areas that are not linearly separable. It is not enough that the measures correlate, but that they correlate well, and it would be beneficial to be able to pick the best out of the ones tested.

The τ is easily interpreted. A τ of 1 consists of only concordant pairs, which are pairs of data points in each of the two datasets with the same ordering. Which is to say that $sign(a_1 - a_2) = sign(b_1 - b_2)$ with $a_1 \neq a_2$. A τ of 0 consist of equal numbers of concordant pairs and discordant pairs. A τ of -1 consists solely of non-concordant pairs. So the taus are

$$\tau_a = \frac{|c| - |d|}{|c| + |d|}$$

$$\tau_b = \frac{|c| - |d|}{\sqrt{(|c| + |d| - |a_{ties}|) \times (|c| + |d| - |b_{ties}|)}}$$

where $|c|$ is the number of concordant pairs, $|d|$ is the number of non-concordant pairs. The measure we used involves a correction for ties, and is called τ_b.

A rank based measure eliminates problems caused by outliers. Extreme values are only one rank away from any other value. Anscombe's quartet(Anscombe, 1973) is a collection of four data sets with very different graphs but the same simple sta-

[3]On the other hand we must use ranking functions that correlate with AP.

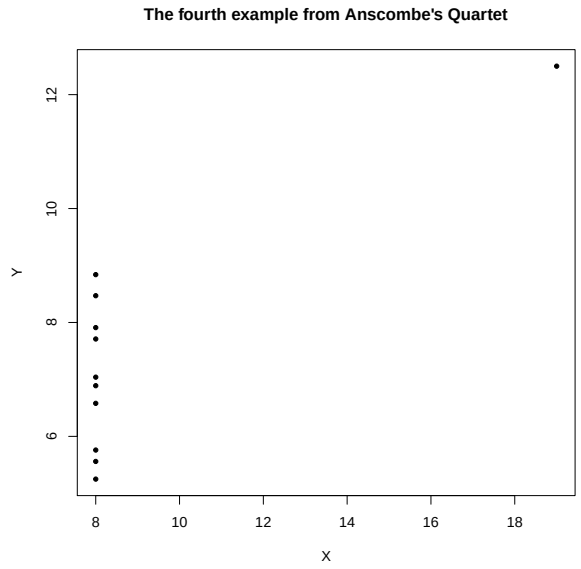

Figure 2.1: The fourth dataset of Anscombe's Quartet. A single outlier causes a Pearson correlation of 0.816

tistical measures. Figure 2.1 is one of these which shows a single outlier can cause some measures of correlation to detect strong correlation where it does not exist.

Some queries appear to be outliers, one query has received a perfect average precision score for every search method we examined, query 461 from the Wiki08 collection. This query is "prison break michael scofield" and only has two documents judged relevant.

2.6 Corpora

We performed our experiments using the collections summarised in Table 2.3. The WSJ collection is a small collection from TREC, consisting of Wall Street Journal articles. For this we use queries 151-200. The larger 2006-8 Wikipedia collection (Denoyer and Gallinari, 2006) from INEX was used for machine learning along with queries from INEX 2006-2008. The even larger 2009 Wikipedia collection with the 2009 and 2010 query sets was also used for validating and testing performance. Both Wikipedia collections consist of English Wikipedia articles.

Corpus	Documents	Average Length	Word count
WSJ	173252	479	218322
Wiki 2008	659387	494	1776759
Wiki 2009	2666190	593	5025214

Table 2.3: Summary of the collections used.

2.7 Conclusion

In this chapter, we examined how performance is measured in Information Retrieval, in the hope that differing retrieval methods may be compared and we can perform machine learning on retrieval.

With these tools we hope to improve stemming. Firstly by learning a stemmer that improves retrieval performance. Secondly by providing a way to improve these decisions with information about the whole collection being searched. Thirdly by predicting when to use stemmed results.

Chapter 3

Learning Stemming with a Genetic Algorithm

3.1 Introduction

Available stemmers all suffer from being optimised towards language correctness. Stemmers like Porter (Porter, 1980), Lovins (Lovins, 1968) and the Paice/Husk stemmer (Paice, 1990) have focused on getting conflation correct. These stemmers do not always improve search engine performance (Harman, 1991)(Hull, 1996).

In addition to learning a stemmer optimised for information retrieval, we would like know how it works. Symbolic learning fits well with the string-rewrite rules some stemmers consist of, and will allow us to examine individual rules and their effect on queries.

A string made up of rule strings seems an ideal representation for learning stemmers with a Genetic Algorithm. The Porter stemmer uses an approximation of the number of syllables in a word as a condition on each rule. We can represent both the rules and a minimum syllable count in a string. A word must have more syllables than the minimum syllable count of a rule if the rule is to match.

Using a judged set of queries we can determine how well stemmers affect the performance of our search engine on a document collection. The judgements provide us with the documents that should be in the results, and allow us to quantify a stemmer's fitness on those queries. This is the Cranfield Paradigm of IR experimentation (Voorhees, 2002). In which we use judged relevance to optimise a measure like Mean Average Precision. Such performance should be compared with the performance of the search engine without stemming, as there is no other way of

comparing values. MAP scores between query sets should also not be used to judge different methods of search as a search method may give different MAP values for different query sets.

Our goal is to have a stemmer which performs well on collections with topics and dialect similar to the training collection and that handles different queries. To do well on data outside the training set, we must avoid overfitting. This occurs when we learn a model that matches noise in the training set instead of matching information that would make a good model. To avoid this, we validate our results by using three sets of data. We choose one set to train, the largest set of queries, the combined 2006, 2007, and 2008 INEX Wikipedia queries. We chose the largest set to ensure there were enough cases to be good learning material. The smaller set of 2009 INEX Wikipedia queries is used to select an individual from the previous learning. This validation is to ensure we choose a stemmer that has learnt enough to be effective on other queries. Finally to accurately report the results we use a separate test set. This prevents us from showing results that would not be expected to repeat on additional queries. For this set, we use the 2010 INEX Wikipedia queries. This involves two different document collections. Both are based on the Wikipedia, but the collection used for the validation and test set is much larger.

We compare performance against other stemmers, as well as the performance of the search engine without stemming. Further comparison of performance on individual queries gives more detail on how stemmers affect each query. Some queries are outliers. An example of this is in Figure 3.1, in which we see how the Porter Stemmer affects individual queries in the 2006-8 set.

Finally, we use paired t-tests to determine if any significant difference exists between the average performance of the learnt stemmer and other retrieval methods. Having a significant improvement on no stemming should show our learning has been successful, but beating the other stemmers would be better.

This method should extend to similar human languages with word variants formed by suffixes, such as Spanish. Changes could be made to deal with prefixes as well.

It is important to compare performance on individual queries. Not all queries are improved by the Porter stemmer (shown in Figure 3.1), and a visual comparison of AP per query will be useful in comparing performance. This highlights the trade-offs made in achieving any improvement to performance.

Stemming strength is useful for gauging how much conflation a stemmer cre-

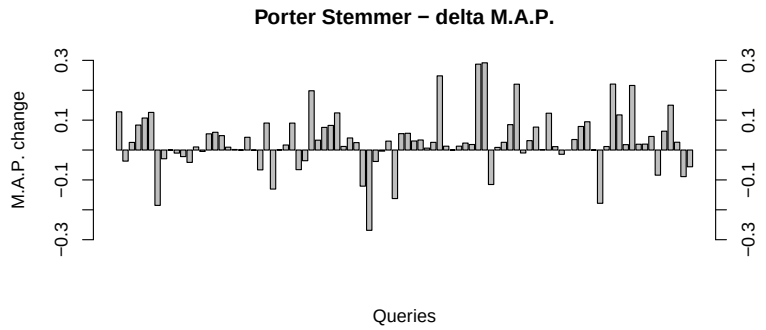

Figure 3.1: Change in M.A.P. per query using the Porter stemmer versus no stemming. Using the 2006-8 set of queries for the 2007 Wikipedia collection.

ates. By measuring the size of groups of terms that are considered equivalent, we can measure how strong a stemmer is. Stemmers are strong if they conflate a large number of terms within a collection.

3.2 Our Genetic Algorithm

To perform learning we need to choose parameters for the algorithm, a fitness function to evaluate individuals, and data sets to train on, validate our solutions and test performance. We also need a representation of solutions, and genetic operators to manipulate the representation during learning. Finally there is the question of what rate to perform each operator, what population size to use and how long to continue each run. We investigate each parameter before final learning.

3.2.1 The Fitness Function

Our fitness is the Mean Average Precision of the results of retrieval on the training set queries using the individual stemmer. This will reward finding additional relevant documents, and moving relevant documents up the returned ranked list. MAP will also penalise shifting relevant documents down the list. It is certainly possible for stemming to hurt retrieval performance. Since stemming improves recall, when it does hurt performance, it is because it has moved relevant documents down the results list.

3.2.2 Data Sets

For training we used the 2006, 2007 and 2008 queries on the INEX 2008 Wikipedia collection (Denoyer and Gallinari, 2006). This set is the largest and should give us more information to train with. We validated using the INEX 2009 Wikipedia collection with the 2009 set of queries. Results were tested on the same collection with the 2010 set of queries. These sets are summarised in Table 3.1.

Words shorter than three characters were removed. These words were too short to be affected by the stemmers represented by the Genetic Algorithm. Words that occur in more than half the documents were also removed. These are bad at discriminating between documents, and create unnecessary work for the search engine. We cut off at half the documents since at that point the inverse document frequency would start down weighting documents with those words. This is not what we want, so we strip these words as well.

	Query set	Query Count	Corpus	Documents
Training	INEX 2006-8	291	Wikipedia 2008	659387
Validation	INEX 2009	68	Wikipedia 2009	2666190
Test	INEX 2010	52	Wikipedia 2009	2666190

Table 3.1: Summary of the data sets.

Table 3.2: Parameters used for the final Genetic Algorithm learning

Population Size	150
Max Generations	200
Representation	Variable length list of rules containing a measure, or separator, and two 4 character strings.
Max Size	60 rules
Min Size	10 rules
Crossover Rate	0.9
Mutation Rate	0.1
Tournament Size	2

3.2.3 Obtaining Fitness

The largest portion of time spent in a Genetic Algorithm is often evaluating the fitness function. We do this for every individual. For our work it comprises searching, ranking and then evaluating the results of a search engine on a large query set. In this project this involves the task of dynamically stemming each query at run time. Typically this work is done once while creating an inverted index and stemming places little additional cost on retrieval. For efficiency's sake, we scan down the index in alphabetical order, starting at the first word that could be stemmed to a query term, and ending when words can no longer be matched. We guarantee that the first two characters of any term will not be affected by the stemming. This prevents pathological cases where words stem to single letters.

Since we spend the vast majority of learning evaluating the fitness function we approximated learning time with the number of individuals evaluated.

3.2.4 Selection

These experiments use tournament selection and elitism. Tournament selection was chosen to provide constant selection pressure. This pressure moves the search towards fitter areas of the search space.

Tournament selection

Tournament selection selects a number of individuals at random with replacement. The number chosen is the size of the tournament. In all experiments in this thesis we use a tournament size of two. This tournament size was chosen as it is almost the only value used in the literature. The individual in the tournament with the highest fitness is selected by the tournament. That individual then gets used by one of the genetic operators to create one of the next population. Using replacement allows the selection of the least fit individual.

Elitism

Without intervention there is nothing preventing the GA from losing the best solution. We use elitism to prevent the loss of the best n solutions from the population. We used elitism with $n = 1$. As with the tournament size it is the default choice. Each generation, the algorithm copies the best individual into the next generation.

This ensures that the information in the best individual is available for further improvement.

3.2.5 The Representation

Our representation is a variable length string of rules. Each rule is fixed in length. Both types of rule are shown in Figure 3.2. The order of rules within the stemmer is important. Rules are grouped by rule separators, and the rules in a group are applied in order. Only the first matching rule in a group will be used.

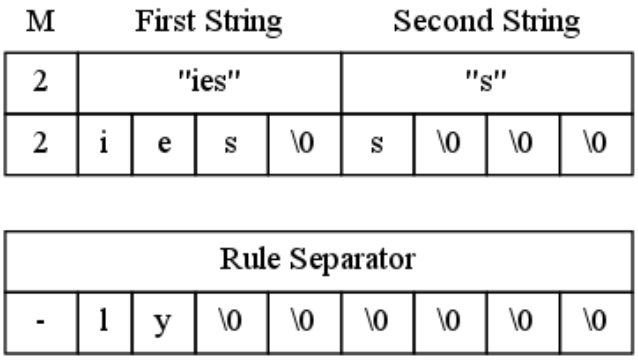

Figure 3.2: Example of both kinds of rule used in the representation of the Genetic Algorithm. When a Rule Separator occurs the strings are ignored.

Table 3.3: An approximation of the first rules from Porter in our representation. Porter is given in Appendix B.

0	'sses'	'ss'
0	'ies'	'i'
0	'ss'	'ss'
0	's'	''
-	''	''
1	'eed'	'ee'
1	'ed'	''

Measure

Shorter words do not have suffixes. If we ignored the length of words we will get errors such as stemming 'is' and 'I' because '-s' is a common suffix. The measure of a term, used by the Porter stemmer, determines if a term is long enough for a rule to be used. Each rule has a number associated. Up to five was deemed sufficient as this is the largest used in Porter, and its purpose is to prevent rules from making terms too short. We only apply rules if the measure of a term is greater than or equal to the number of the rule.

A run of at least one vowel is V, and a run of at least one constant is C. We treat 'y' as a vowel if the character after it is not a vowel or the 'y' is the last character. Otherwise it is a consonant. The C and V at the ends are optional. Measure counts the number of times a group of VC occurs.

$$[C]\{VC\}^n[V]$$

$$measure = n$$

Separating Rules

There are many words with more than one suffix. In order to deal with all the 'possibilities' we allow for more than one rule to be applied to a word. The other stemmers group rules together. The first matching rule of any group is applied. Having multiple groups allows for multiple rules to match and remove suffixes.

Rather than introducing additional complexity into our representation, we took advantage of the fact that rules could be grouped by splitting them. A '-' character in the place of the measure turns a rule into a separator between groups. After any rule has succeeded in replacing a suffix, other rules are skipped until a separator is found, indicating a new group of rules. This is similar to both the Porter and Lovins stemmers. The strings in a rule separator are not used for matching or replacing suffixes, but exist to prevent it from being a special case for crossover and mutation.

Implementation

Normally stemmers would be used when building the index. Since indexing is slow, and we do not know what terms should be conflated until after learning we use an index of raw terms. Stemming has to occur dynamically during learning. For any individual we then have to stem all the terms in the query. Then for each term in the

index we stem, and if it matches a query term they are conflated. To speed up the process we only consider terms in the index that match the first three characters of a query term. Stemmers are unable to alter the first three characters of a term.

The rules already found in stemmers are very close to a suitable representation. We only needed to choose limits, such as matching suffixes of up to four characters, and to simplify the many conditions placed on rules.

The representation of our rules consists of a single character and two suffixes. The character differentiates between rule separators, and normal rules. Normal rules use this space to store the minimum measure of a matching word. This is the same measure that Porter uses. Suffixes within a rule describe the suffix that is matched and the replacement suffix. While the former must always have a suffix, the latter can consist of nothing, and the rule will remove the matched suffix. Separator rules segment the other rules into sections from which only the first matching rule is used. This allows rules to prevent the use of alternate rules if they get used first.

The first example rule in Figure 3.2 will replace an 'ies' suffix with an 's' suffix if the word has measure of 2 or more.

The entire string representing an individual has a variable length. A size limit of 60 rules is in place to prevent stemmers becoming too big. When using a variable string representation with Genetic Algorithms the individuals tend to grow during learning, without an increase in performance. Crossover maintains the string size by only splitting individuals between the strings. The strings are treated as immutable. Allowing the mixing of parts of strings would ruin the work taken to ensure they are suffixes found in the collection. Mutation also maintains the number of rules.

3.2.6 Generating Suffixes

By generating suffixes carefully we can decrease the search space, and speed up learning. If a suffix is not found in the document collection then it cannot affect the fitness of a stemmer, and we should ignore it.

Once we checked the distribution of suffixes in the vocabulary, we found many are rare or unique and could also be ignored. We would have little information as to how fit rules with those suffixes are, and it is unlikely they would give us any improvements. Finally, we ignore endings of short terms. These are terms of four letters or less. Such endings are unlikely to be actual suffixes, and in the extreme case it will be the entire word if the term is too short.

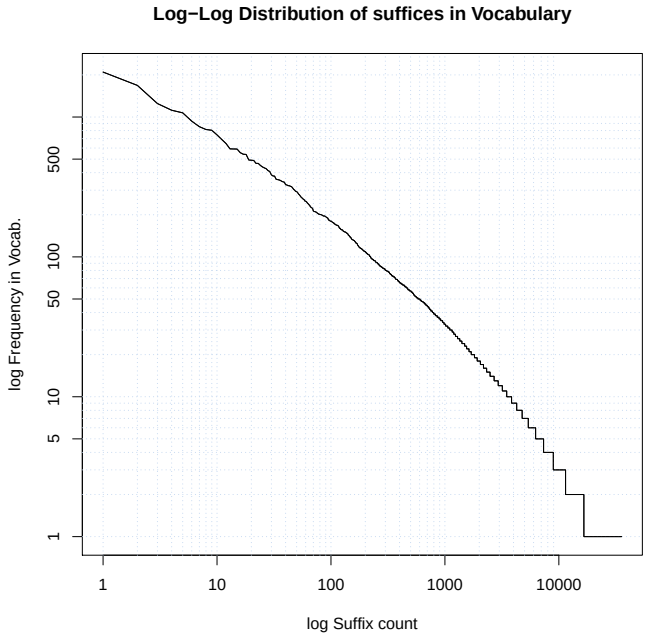

Figure 3.3: The frequencies of unique 4 letter suffixes. Each word is counted once.

The distribution of suffixes in Figure 3.3 is similar to that of Zipf's law. This law also describes the distribution of how words occur. This shows there will be many suffixes that are unique or very rare. So removing them is worthwhile to focus search on more useful suffixes.

To generate a suffix for a rule, we randomly select a word from the trimmed vocabulary of the collection. In the vocabulary, each unique word occurs only once. We choose up to four characters from the end of that word. If we are generating a suffix for the second string, zero characters is a possibility to allow for suffix removal.

3.2.7 Genetic Operators

We chose to treat the substrings in the rules as atomic. This reduces the time spent finding good strings to use. Crossover and mutation treat them as a choice from all the suffixes in our list. This list is all possible suffixes in the vocabulary except unique ones.

Crossover

Crossover for this representation is allowed between the sections of the rules. This maintains each string as one taken from the vocabulary. Splitting up words that are guaranteed to exist in the collection to some other string which may or may not is unlikely to help learning. Otherwise the crossover is a standard one point combination of two parents, the point in the second parent chosen to prevent the individual from going over the maximum size. This is necessary due to the propensity for variable-length individuals in genetic algorithms to steadily increase in size.

We examine crossover and mutation rates experimentally. Our search is focused on higher crossover rates because this is what the literature has done. In particular, Koza (1992) uses crossover rates as high as 80% or 90%. A reason for this is that recombination via crossover is where most of the learning is done, and mutation maintains diversity and new parts to combine.

Mutation

When a population is undergoing a lot of crossover, there is a danger of the population losing some subsolutions. Mutation is an operator designed to maintain diversity in the population by randomly altering some of the individuals. For us this prevents the loss of suffixes from the population. Mutation allows for the generation of additional strings into the population.

Mutation is used to keep the solutions within the population diverse. For learning stemmers, diversity will be a problem for the strings in rules. The number of strings that can be in the population is small, but the number of possible strings is large. So mutation aids in exploring further strings as learning occurs. We rely on crossover to recombine rules in different orders.

Mutation creates a child from a parent individual by altering a single part of a rule in the parent. This may change the measure of a rule, or even change a rule in to a rule separator. If a string is to be mutated then a new one is generated in the same manner as the other was initially. So only the second rule may be blank.

3.3 Experiments

3.3.1 Population Size

We care about the speed and quality of learning. Finding a good population size will lead to both. Before experiments can be run, we must find values for the learning parameters for the GA to use. We need to find a number of generations to examine before stopping, a population size and the number of times we should repeat learning. Population size determines how many individuals make up each generation of the Genetic Algorithm. The optimal value been shown to be problem-specific. The number of individuals is balanced between providing information about the search space, and processing time required.

To find an appropriate value for population size, experiments need to be performed. We optimise this to maximise fitness and minimise the number of evaluations. Evaluations of stemmers take time, and fitness is our measure of success. The best population size is one that finds fit stemmers in a short amount of time.

Fitness should increase, and level out as learning progresses. This information will also allow us to identify the number of runs necessary and how long they should run before we can be sure a good stemmer has been learnt. Of course learning might not be successful, given the stochastic nature of the Genetic Algorithm. In this case, starting another run is better than continuing with a population that has been unsuccessful.

Our baselines are no stemming, the S stemmer, the Porter stemmer and the Lovins stemmer. Their performance is shown in Table 3.6.

We will examine the behaviour of population sizes from 50 to 200, in increments of 50. The maximum number of generations will be 200. This will ensure we have enough information to correctly make a decision about better parameters. Beyond this amount single runs of the genetic algorithm take more than 7 days. A single query will take around 7ms, and the query set consists of 391 queries. A single run consists of 200 generations of 200 individuals each. Results for each parameter value are the average of 20 runs. Multiple machines were used in these experiments.

The number of individuals examined is used as an approximation of the time taken to learn. With this we may use the number of individuals until a certain level of performance is reached as a simple indicator of how well learning is occurring. The performance of the Porter stemmer was selected as it was the highest baseline. Other levels of performance were too easily met. Using other baselines is difficult as

even random selection in the first generation produces stemmers which improve on no stemming. We still need to take into account the maximum amount of learning possible, but speed is important given the length of time these experiments take.

Results

As our baseline we use the performance of the Porter Stemmer, which on this query set has an M.A.P. of 0.354. Not all runs of population size 50 and 100 reached this level of performance. To calculate averages we assumed that these would reach the baseline during the next generation. The average performance of the best individual is given in Table 3.4. Values for the average generations and individuals examined are also in this table.

As seen in Figure 3.4 and Figure 3.5 it would be misleading to only examine performance per generation. Per individual, a population size of 150 beats that of 200. This is the important measure as it is individuals examined that approximates time spent, not generations examined. Obviously a generation of 200 individuals will usually take more time to evaluated than a generation of 150 individuals. Moreover, the performance increase from 150 to 200 is small.

Table 3.4: Performance of different population sizes. All numbers are means over all runs.

Population Size	Best MAP	Generations to Porter	Individuals to Porter
50	0.356	149.4	7470
100	0.358	88.2	8820
150	0.359	26.7	4005
200	0.359	20.4	4080

Conclusions

From this experiment it is clear we should use a population size of 150 individuals. Given the small difference between that and 200, we shall trade-off individual run performance for the ability to perform more runs in the same time. The time taken to run experiments is proportional to the number of individuals examined, and having a population size of 150 rather than 200 will allow 33% more experiments. Since

Genetic Algorithms can be sensitive to the initial population we choose more runs rather than more individuals per run.

3.3.2 Crossover and Mutation

Another chance to optimise learning is find a good proportion of crossover and mutation to use to generate child populations. The proportion of reproduction is derived from these two rates. All three rates must create a new population of the right size, so the rates must sum to 100%.

We seek to search in an area of the space already known to do well for Genetic Algorithms. Especially in this case, where a large part of the problem is finding the optimal combination of rules to use. That is what crossover does, so we expect the ideal crossover rate to be high, with smaller values for mutation and reproduction.

Since there are more combinations to examine, we limit the learning to 100 generations. This limitation is pragmatic and aims at reducing the amount of time required. We only need to find the parameters that perform the best, and do actual learning in later experiments. Results for each combination of parameter values is the average of 20 runs.

Table 3.5: Final average Mean Average Precision over Crossover and Mutation Rates.

		Mutation Rate		
		0%	10%	20%
Crossover Rate	70%	0.3573	0.3587	0.3577
	80%	0.3561	0.3577	0.3592
	90%	0.3573	**0.3595**	N/A

Results

As these runs were consistent in regards to how fast they reached Porter levels of performance, we examined the final fitness of each group of runs. The final fitnesses are presented in Table 3.5. A Crossover rate of 90% and Mutation rate of 10% seems to give the best trade off between maximum performance and learning speed, obtaining the highest average final fitness. This may be seen in Figure 3.6, where it gets

the best maximum fitness. Apart from that and 80% crossover with 20% mutation, the rest are abysmal. Table 3.5 confirms that it reaches the highest average fitness.

This setting gave the best average Mean Average Precision at the end of 100 generations. Having no room left for reproduction, this will not keep old individuals in the population, except for the best one which is kept by elitism.

3.3.3 Refining the representation

Both the use of measure and rule separators seem like things that could help an individual stemmer to improve performance, but if the Genetic Algorithm does not take advantage of them then they may hurt learning by increasing the search space. Therefore we will investigate the use of both, by testing all combinations of their use, and comparing how they affect learning and final performance of stemmers.

Measure

While Porter uses measure the individuals created using the Genetic Algorithm might not take advantage of it. To determine if the use of measure is worthwhile we will compare two sets of runs, one using measure and one ignoring the measure values. One concern is that measure may increase the difficulty of finding good rules.

Rule Separator

Rule separators are a feature of rule-based stemmers. We will test if including them is worthwhile when learning with a Genetic Algorithm. They should allow for rules to cooperate, and improve fitness.

Separators are rules which do not perform any string transformation, instead they split rules into groups. The rules between separators become groups. Each rule in a group is tried in sequence and only the transformation will be performed.

Our version of the S stemmer, given in Table 2.1, uses a single group to test longer suffixes before smaller suffixes. Since it is an effective stemmer that lacks any separation between rules it gives us reason to doubt the efficacy of separating rules.

Experiment

We wish to compare learning behaviour of representations with and without both measure and rule separators. Using all four combinations, we judge the learning of each by the maximum fitness reached. Results for each setting are averaged over 20 runs.

Our ultimate aim is to have a simple representation. If either part of the representation is detrimental to learning we should remove it. If there is a no effect or an improvement on learning then we will keep them. Removing either the separators or the measure from a stemmer is easy, adding them back would be another learning task.

Results

The results are presented in Figure 3.7, which shows that not being able to separate rules has a detrimental effect upon learning. In both cases where it is left out (the left two graphs) there is visible decrease in final fitness. Measure appears to have little effect on performance.

3.3.4 Final Learning

With a crossover rate of 90% and mutation rate of 10%, a population size of 150 individuals and a representation including both rule separators and measure, we could begin learning a final product.

The actual learning of a new stemmer used 54 runs. These were trained on the 2006-8 INEX Wikipedia collection. The parameters used are specified in Table 3.2 There are far more stemmers than actual runs because we use the best unique individuals from every generation in a run. This is due to the varied performance as the Genetic Algorithm continues. The best stemmers in a generation (on the validation set) were almost never from the last generation, suggesting overfitting.

Results

Figure 3.8 shows all the stemmers examined that were better than the baselines in Table 3.6. The overfitted individuals are those which perform well on the training set, and poorly on the validation set. In Figure 3.8 these are on the bottom right. Our chosen stemmer is at the top, somewhat to the right. Performance of this stemmer

on all query sets is given at the bottom of Table 3.6. Rules for this stemmer are given in Table 3.7. Those rules are a transcription of the raw rules, as can be seen from the rule separators at the end.

On the test set the performance of the final stemmer provides evidence for a significant improvement over no stemming and the Lovins stemmer, with the t-test p-values of 0.0005 and 0.0002 respectively, but no significant improvement over Porter or the S Stripper. A detailed comparison of precision at different values of recall is given in Figure 3.9. This figure shows the proportion of relevant documents in the results (Precision) as the results include more relevant documents (and recall increases). The leftmost points correspond to the most important documents at the start of the ranked lists. Here we see our stemmer performs the best where it matters most, and all stemmers examined beat no stemming at every point of recall.

Table 3.6: Average Performance of Stemmers on all Query Sets

Stemmer	MAP training set 2006-8	MAP validation set 2009	MAP test set 2010
None	0.327	0.343	0.351
S Stripper	0.348	0.365	0.381
Porter	0.354	0.347	0.380
Lovins	0.326	0.321	0.297
Ours	0.357	0.369	0.386

The best stemmer found during validation is listed in Table 3.7.

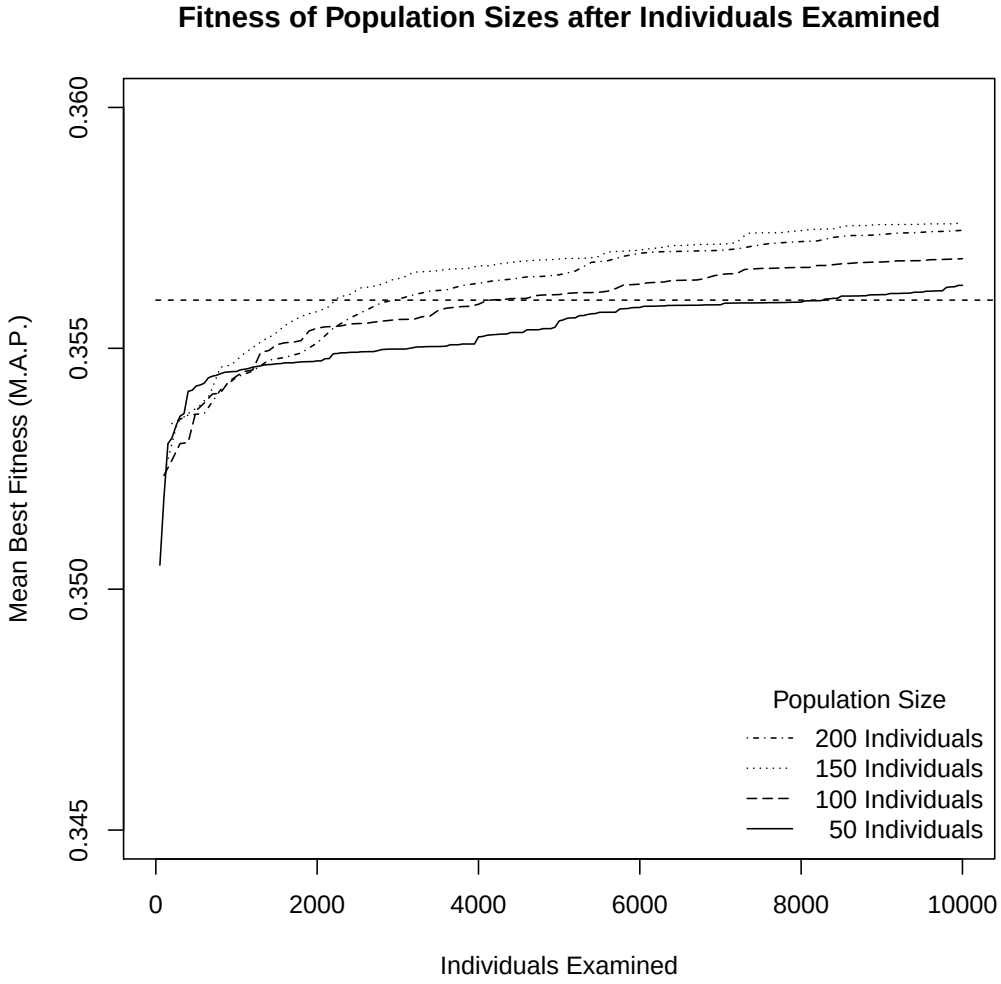

Figure 3.4: Average fitness against number of individuals examined for population sizes. The horizontal dashed line is the Porter baseline.

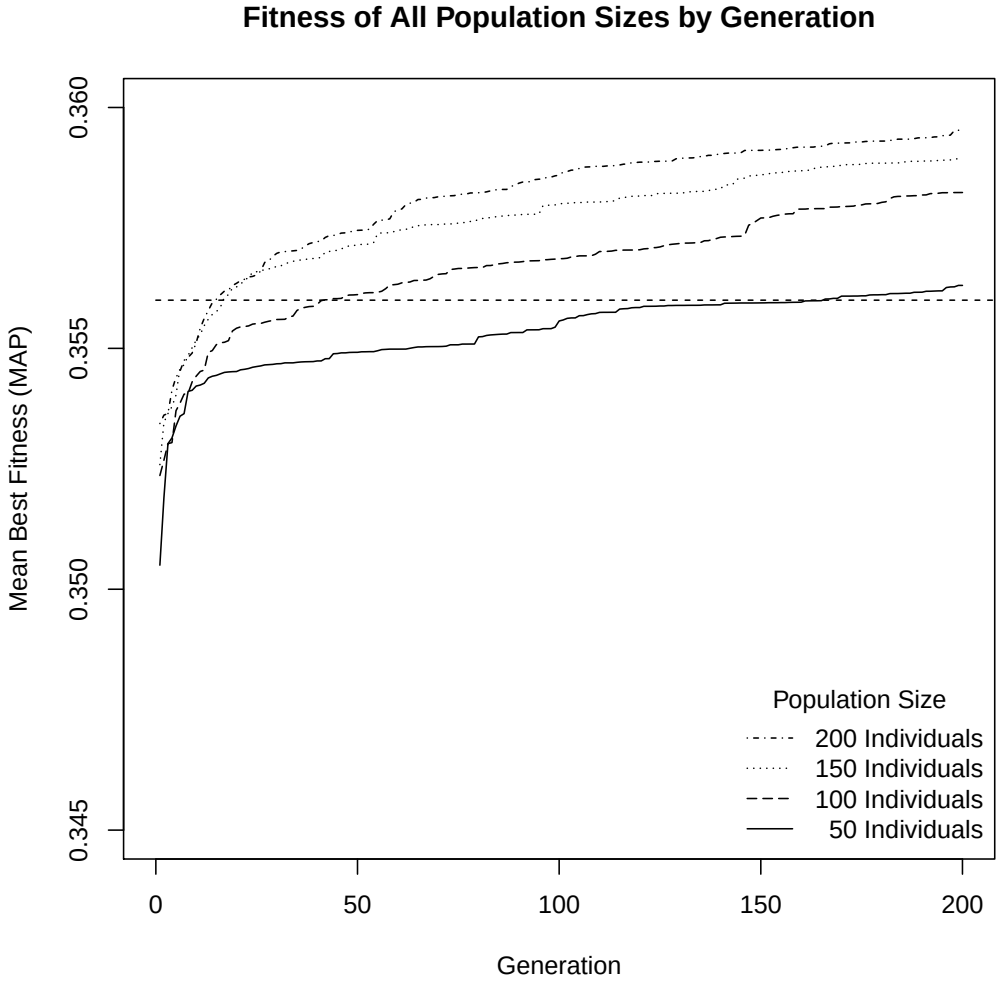

Figure 3.5: Average fitness against generations for population sizes. The horizontal dashed line is the Porter baseline.

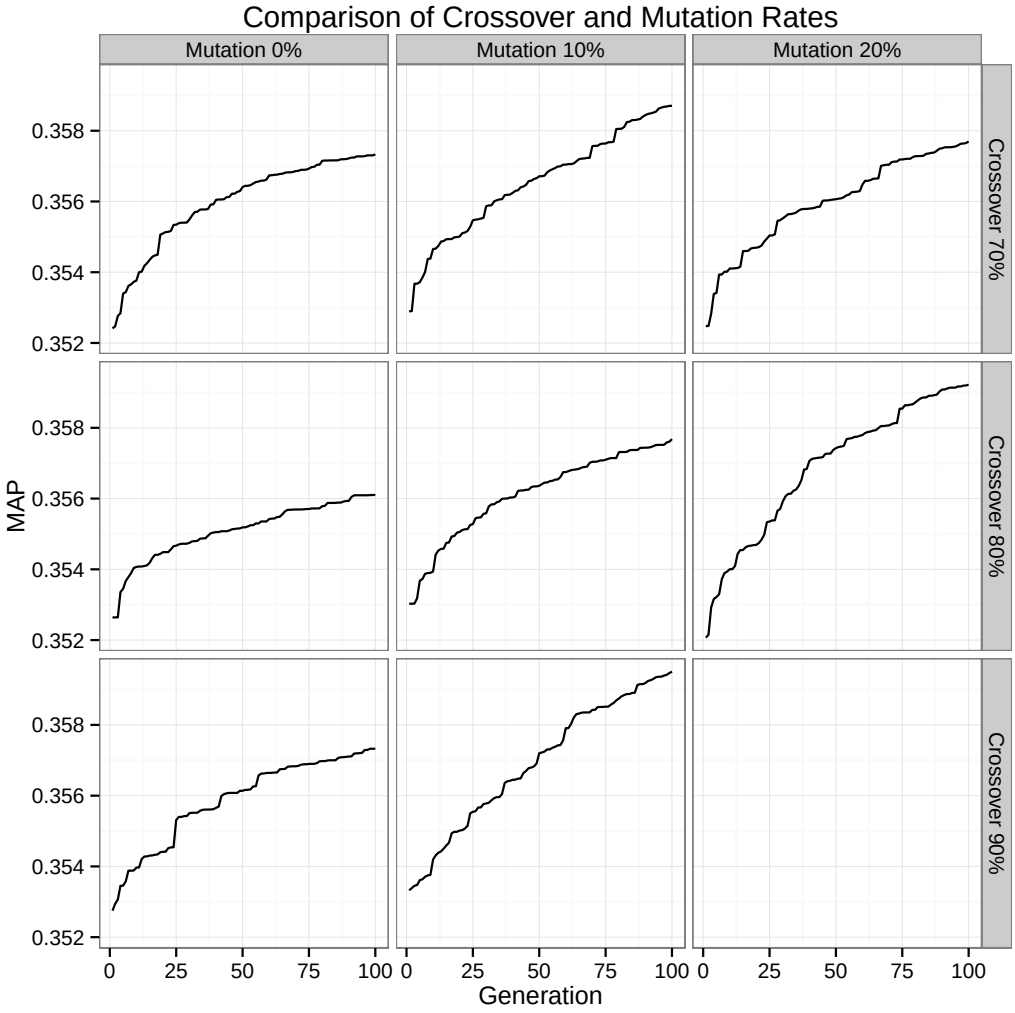

Figure 3.6: Average fitness over generations for different crossover and mutation rates. Population is fixed at 150 individuals.

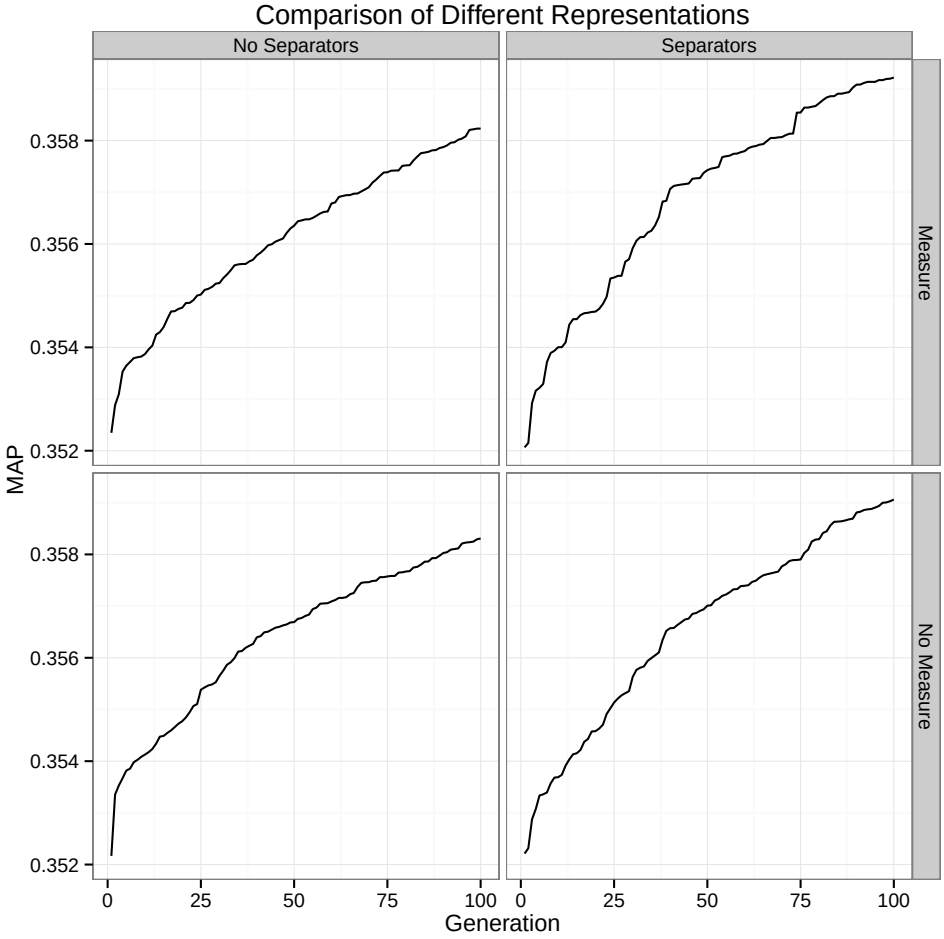

Figure 3.7: Comparing learning using a representation with and without measure and rule separators or measure. Learning has a population of 150 individuals, a crossover rate of 0.9, and a mutation rate of 0.1

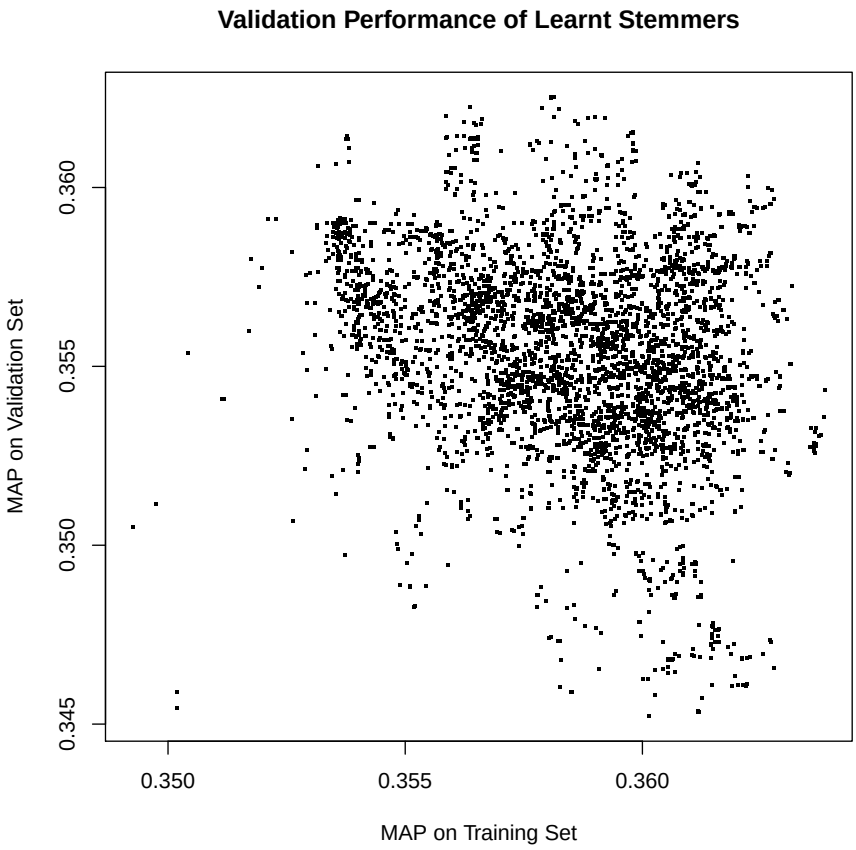

Figure 3.8: Comparing performance on the training set and the validation set. The stemmer selected by validation is the topmost point.

Table 3.7: Learnt Stemmer

4	ya	t
4	ya	t
2	asse	
1	s	
0	aris	osus
4	er	ka
4	ed	ey
4	ie	ames
3	aris	osus
4	dal	ey
4	ie	a
3	atv	t
4	t	ins
1	oku	ammi
3	ki	hoek
0	rton	g
0	es	
2	uppe	n
4	ely	

3	ss	ds
3	ak	
2	d	
1	b	wr
2	as	o
2	uppe	
4	ely	

3	ss	ds
0	ak	
2	d	
1	b	ion
2	as	o
2	ze	
0	ure	ites
0	ak	oids
2	d	aust
3	ss	
3	ak	oids
3	d	atul
1	b	wr
2	as	o
0	ta	ing

3	ss	
3	a	oids
3	jo	aust
0	b	wr
2	as	o
0	tang	ing

3	ez	ont
3	n	
3	ng	nder
2	ist	
0	ure	ites
0	an	
1	an	
2	l	wsie

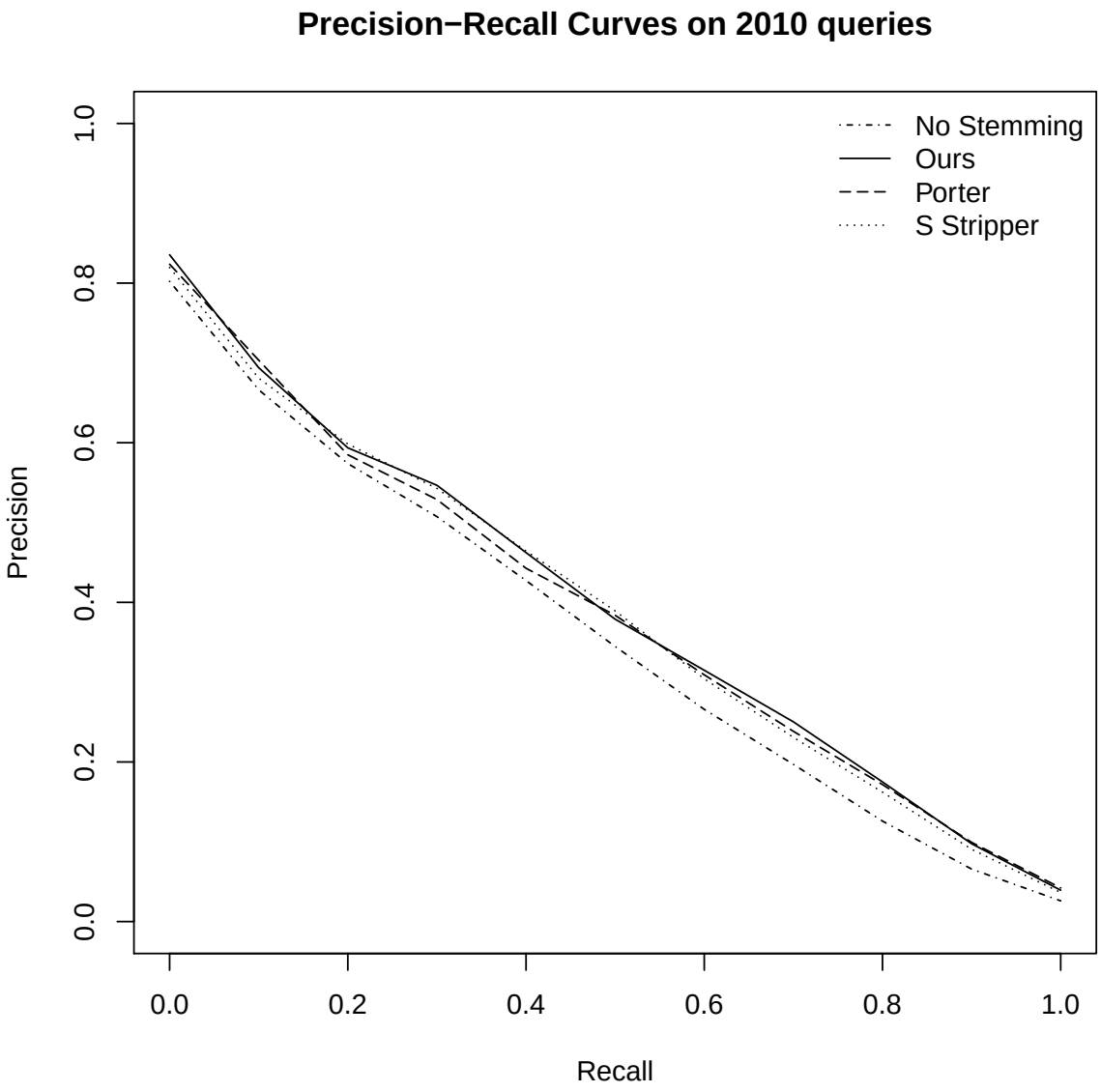

Figure 3.9: Precision at different values of recall on the test set, 2010 Wikipedia.

3.4 Simplifying Learnt Stemmers

Genetic Algorithms have a tendency to generate solutions that contain extraneous parts that have little impact on fitness. We made an effort to remove these useless rules to aid understanding of how the rules work, and also to prevent mishaps on unseen queries. Stemmers are capable of decreasing the performance of queries as shown in Figure 5.1 and Figure 5.2.

Table 3.8: Simplified Learnt Stemmer

1	s	
2	d	
0	ak	
2	d	
2	ze	
0	ta	ing
3	a	oids
3	n	
2	ist	
1	an	
2	l	wsie

This was done using a greedy approach. The first rule found that could be removed without negatively impacting performance was removed. As this is a form of learning, we use the training set. This was repeated until no rule could be removed without hurting performance. When then tested again on the validation and test sets to ensure that no performance had been lost. No performance was lost on any of the training, validation or test set. In fact, a slight (and insignificant) improvement occured on the validation set. The rules are much easier to understand. See the sheer number of rules in Table 3.7 that are removed in Table 3.8.

The 's' removal rule accounts for a large amount of the performance of the stemmer. Most of the conflation occurs due to this rule. Note that the 's' rule occurs if the term has at least one measure, which is the approximation of syllable count from the Porter Stemmer. This will be most words.

The first two rules give equivalence classes like 'nobel', and 'nobels'; and 'prize', 'prized', and 'prizes' which improves the performance on that query. Some rules

appear to fix missing spaces, where a space is missing between a query word and the next word. Learning will only discover this if the next word is very common, and it occurs enough for 'bestand' to conflate with 'best.' The '-and' suffix is removed by a combination of the rules to remove '-d' and then '-an.' This will be prevented for long words where the rule to remove '-n' will happen before the '-an' rule. Another example of a common word being removed from the end is the removal of '-a.'

The last rule is odd. It is never used to conflate two terms. No word uses it that ends up matching a query term. Removing it only slightly affects performance. We believe it is used to prevent words from being conflated. Words which have not used a rule in the last section that end in '-l' will not match words that end in '-l' after the other rules in that section.

Finally, we measured the average number of words stemmed together by a number of stemmers, including ours. The results are in Table 3.9. Our stemmer is only slightly stronger than the S Stemmer, and weaker than the others. Comparing this with the performance in Table 3.6 we can see that strong stemmers are not always effective. We make no claims about suitability for other tasks like clustering, but strong stemmers are risky for search engines.

Table 3.9: Stemmer Strength

Stemmer	Average Equivalence Class
S Stemmer	2.1
Ours	3.4
Porter	9.0
Lovins	24.3
Paice-Husk	42.0

3.5 Conclusion

In the work of this chapter, we found good parameters to provide a Genetic Algorithm for the task of learning a stemmer. With those we were able to learn a stemmer that performed comparably to other stemmers. Our stemmer had rules that were artifacts of using a Genetic Algorithm. With a greedy form of pruning, these were removed and the stemmer became simpler. The result is a short, understand-

able stemmer that is focussed on Information Retrieval performance. This process should be repeatable on other collections and queries in human languages with a similar suffix structure to English.

Chapter 4

Refining Stemmers

In the last chapter we created a rule-based stemmer. In this chapter we try to improve the decisions made by stemmers using information from the document collection.

Some stemming rules work well on most words but have errors on difficult words. One such case is conflating 'treats' and 'treaty.' Normally removing '-s' and '-y' might improve performance, but in this case 'treats' has another meaning that may hurt performance. This occurs because stemming rules look solely at the characters of the words. There is a document collection full of information that is ignored by the rules. We investigated how to use this information to prevent good stemming rules from causing errors. Psuedo code is given in Table 4.1.

Given a query term **in** the query:
Stem the query term
For each term **in** index:
    If the index term is the original query term:
        The index term conflates with the query term
    Else, if the stemmed query term and index term match:
        if the two terms are similar
            The index term conflates with the query term
Combine results for those terms

Table 4.1: Pseudocode for improving stemming with collection information

Each group of terms that a stemmer conflates can be seen as a fully connected graph. An example is given for the query 'Aardvarks eat for free' in Figure 4.1.

Terms are vertices and edges connect the terms that a stemmer conflates. We can extend this model by adding weights to the edges. These weights will refine the decisions made by the stemmer. The search engine can stem selectively, conflating terms only when the weight between them is large enough. This is a final test before stemming terms together. We can then extend this by using the weights to alter the term frequencies of conflated terms. This will make the effect of stemmed terms dependent on how similar they are to query terms.

Similarity between connected terms is the obvious candidate for the edge weights. Alternate forms of a word should be similar. Words that are conflated incorrectly, but belong to different concepts should not be similar. There are different measures to determine how similar terms are within a document collection. These use the patterns of how terms occur within documents, which already exist in the index of the search engine. When terms are in the same document we gain evidence that they should be conflated. When terms do not occur together, this fails to provide any evidence that those terms belong to the same concept. Document size may play a part here. It is more likely that any two terms will occur together in a large document than a small one.

By doing this, we add global information from the entire document collection to refine stemming. There is already some work in this area, notably work by Croft and Xu (1995). However that work was restricted to examining a single form of term similarity, the Expected Mutual Information Measure. We will examine more measures, and compare their performance. We also consider weighting the search engine scores by term similarity. This work can be easily applied to other stemmers. We investigate how it affects other stemmers.

Further work exists that creates equivalence classes of words from just term co-occurrences. First done by Xu and Croft (1998). Other work (Peng, Ahmed, Li, and Lu, 2007) uses the rest of the query as the context to make stemming decisions. More recently, after the work conducted in this chapter, Paik, Pal, and Parui (2011) did so again. Both of these works examine using a single measure to partition the entire vocabulary, rather than altering the results of stemmers.

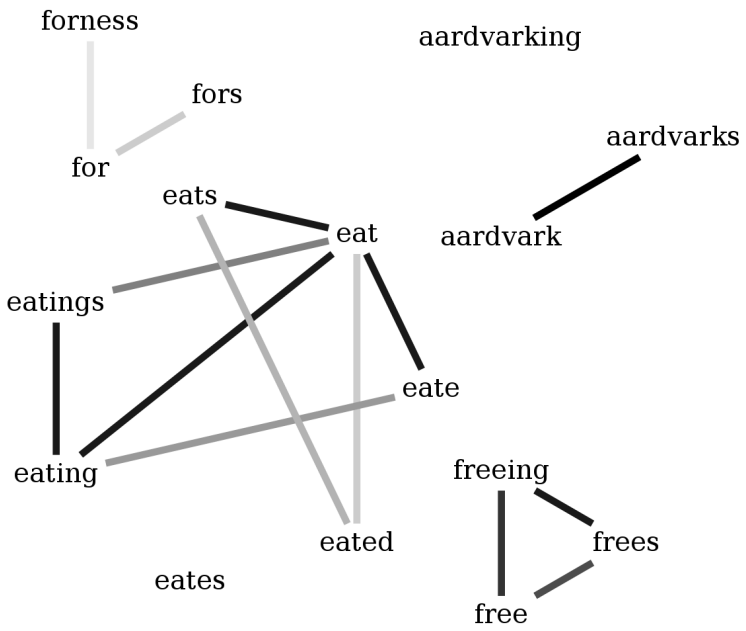

Figure 4.1: Stemming as a weighted graph.

4.1 Measuring Term Similarity

There are many previously published measures we could use, so we tried a few and selected the best. Using a threshold on a measure is much simpler than weighting term frequencies, so we tried that first. Evaluating each measure involves finding a parameter to use as a threshold. Terms that are more similar than the threshold will be stemmed together.

4.1.1 Cosine Similarity

Cosine similarity is the cosine of the angle between vectors created by considering each document as a separate dimension. The length of a vector along a dimension is the tf.idf score for that term and document, the product of the term frequency and the inverse document frequency. This is widely used in Information Retrieval to judge how similar words are. It is based upon the vector space model of information retrieval.

Let t_1, and t_2 be terms, and let $v(t)$ be the vectors of tf.idf scores for every document of term t.

$$Cos(t_1, t_2) = \frac{v(t_1) \cdot v(t_2)}{|v(t_1)||v(t_2)|}$$

4.1.2 Jaccard Index

A simple method using the sets of documents terms occur in. This ignores how often terms occur within documents. The Jaccard Index is the ratio of documents containing both terms to documents containing either of the terms. If term always occur together, this will be one. If they never occur together the Jaccard Index will be zero. As it ignores term frequencies, this measure uses less information.

Let t_1 and t_2 be terms, and let $d(t)$ be the sets of documents containing term t.

$$J(t_1, t_2) = \frac{|d(t_1) \cap d(t_2)|}{|d(t_1) \cup d(t_2)|}$$

4.1.3 Pointwise Mutual Information

Pointwise mutual information is the gain in information about the state of one value, if we know the value of another. This is found by comparing how far the probabilities are from independence. Similar terms should occur together more often than by chance.

Let t_1 and t_2 be terms, and $P(t_1, \ldots, t_k)$ be the estimated probability that a document contains all of $t_1, \ldots, t_k$.

$$PMI(t_1, t_2) = log \frac{P(t_1, t_2)}{P(t_1)P(t_2)}$$

We also use a normalised variant of PMI, from Bouma (2009). This can weight term frequencies. Unmodified PMI gives values outside the zero to one range desired. We clamp negative values to zero to prevent removing term frequency values when words occurrences happen less than the independent probability.

$$nPMI(t_1, t_2) = \max \left(\frac{log \frac{P(t_1, t_2)}{P(t_1)P(t_2)}}{-logP(t_1, t_2)}, 0 \right)$$

4.1.4 Expected Mutual Information Measure

This measures how much information the occurrence of a term provides in predicting the occurrence of another term. This is zero when the occurrences are independent. This is PMI scaled by the actual probability of occurrence.

$$EMIM(t_1, t_2) = P(t_1, t_2) log_2 \left(\frac{P(t_1, t_2)}{P(t_1)P(t_2)} \right)$$

Similar terms should give information about the occurrence of each other. Xu and Croft used a measure based on EMIM (Xu and Croft, 1998).

4.1.5 Kullback-Leibler Divergence.

KL Divergence is a non-symmetrical measure of how well one distribution (P) may be modelled using a code minimised for encoding another distribution (Q). The result is the extra number of bits required to encode P using Q. We use the divergence between the two terms probability distributions of co-occurrences.

KL Divergence is only defined if Q is non-zero when P is non-zero. So we compare the set of documents in which both terms occur against the set of documents P occurs in. Other uses of KL Divergence in IR avoid this issue by using the collection distribution as Q.

Let P be the distribution of documents containing term t_1, Q be the distribution of documents containing t_1 or t_2 (since Q needs to be non-zero when P is non-zero). Let D be the set of documents, and $P(d)$ and $Q(d)$ be the values of those distributions given document d.

$$DL_{KL}(P||Q) = \sum_{d \in D} P(d) log_2 \frac{P(d)}{Q(d)}$$

As this is non-symmetrical (i.e., swapping P and Q may change the value) this would make the graph of stemmed terms directed. This is interesting because it means the particular word form used in the query matters. For example if the stemmer conflates 'fishing' and 'fish', using KL divergence would be able to conflate the words if the user supplies 'fish' but not if they supply 'fishing'. All the other measures are symmetrical.

4.2 Weighting Term Frequencies

We have also tested weighting the term frequencies. Rather than removing the term frequencies from stemmed terms that are not similar enough to the query terms, we weight them according to how similar they are to the original query term. Those that often co-occur with the term itself should be weighted highly, and those that do

not should be weighted less. A sigmoid function is intermediate in shape between a linear function and a threshold function. This also maintains monotonicity; a term that is more similar should always be weighted higher than one that is less similar. Finally, the range of values given by the sigmoid function is between one and zero. These values should limit the behaviour of terms between that of terms treated like the original query term, and terms that are not considered at all.

$$Sigmoid(t) = \frac{1}{(1 + e^{-t})}$$

Adjusted version:

$$f(t; a, b) = \frac{1}{1 + e^{-b(t-a)}}$$

This provides an adjustable sigmoid that ranges between 0 and 1. The b value controls the steepness of the sigmoid. As it increases the sigmoid becomes closer to a sharp threshold. The a value controls the position of the threshold between 0 and 1.

This form of weighting can represent a threshold when b is sufficiently large, and so can do no worse than using a threshold. If weighting term frequencies turns out to do no better, then using a threshold provides benefits of simplicity and efficiency.[1]

We will only examine weighting using the best measure found using a threshold. We will also restrict the range of a to around the threshold found for the previous experiment.

4.3 Parameter Search

To threshold based on similarity we need to know at which values of similarity we should conflate. We use a single parameter as a similarity requirement on potentially stemmed words. Words must be more similar than the parameter before we conflate them. These experiments will find a good parameter to use for each measure.

There are two widely used and simple approaches to parameter search - grid search at some granularity, and binary search. Binary search is exploitative; it focusses on areas previously found to be good. Grid search is explorative, but not exploitative; it may be guaranteed to find a good value, but it will not necessarily find the best, even if it searches near it.

[1] Allowing one to use integer values for example.

We combine the two, starting with grid search at a large granularity, 11 points covering the entire range. The search will then focus around the best point, and cover the range of the 2 points either side. We continue until the neighbouring points of the best have the same value. The centre will be the final parameter.

As the two dimensional parameter search is much more intensive and weighting term frequencies is slower than using a threshold, we only test weighting term frequencies upon the best measure.

4.4 Experiments

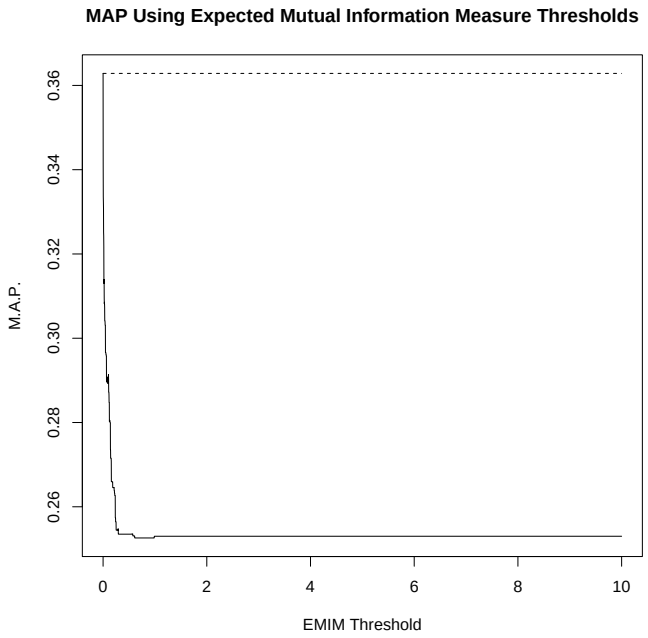

Figure 4.2: Parameter search for a Expected Mutual Information Measure threshold

4.4.1 Term Similarity Thresholding

For each form of term similarity, a parameter was found to use as a threshold that gave the best results on the training set. The best is then evaluated on the test set. There is no validation set for this as there is only one result from training and therefore nothing to compare on a validation set. For training we use the queries on the

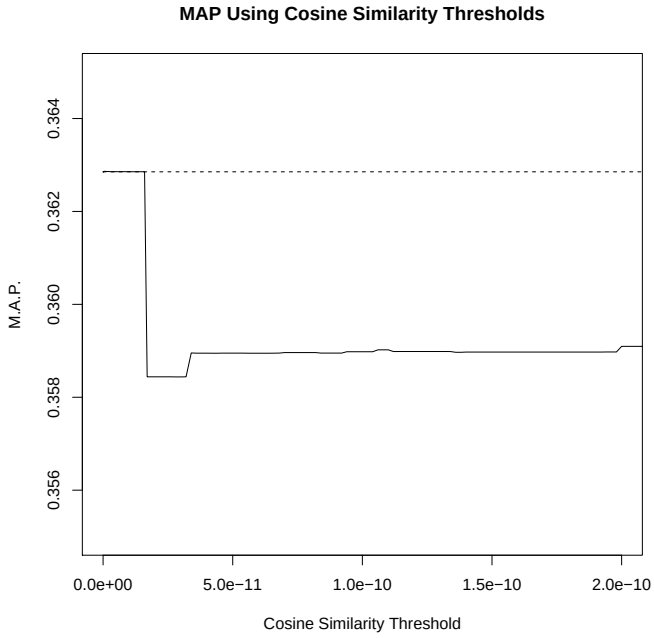

Figure 4.3: Parameter search for a Cosine Similarity threshold

2008 Wikipedia Collection and test on the 2009 Wikipedia Collection.

The parameter search is done by starting with a grid search over the range of possible values, and optimising performance as measured by Mean Average Precision.

The baseline performance is the MAP of the stemmer we are trying to improve. Gains might seem small, but are larger if considered as an improvement upon the improvement already gained by using a stemmer.

The stemmer used is Porter as it improves performance and conflates more terms than ours as seen in Table 3.9. A stronger stemmer provides more opportunity for correction using collection based information. Porter provides us with three times as much conflation as our stemmer does. It is also a good stemmer to improve, as it performs well. Improving a stemmer like Lovins, while even stronger than Porter, may not prove anything. We examine performance on our stemmer after parameters have been found.

Results

The threshold values found are listed in Table 4.2. Except for EMIM in Figure 4.2, all measures were able to increase performance on the validation set. PMI gave

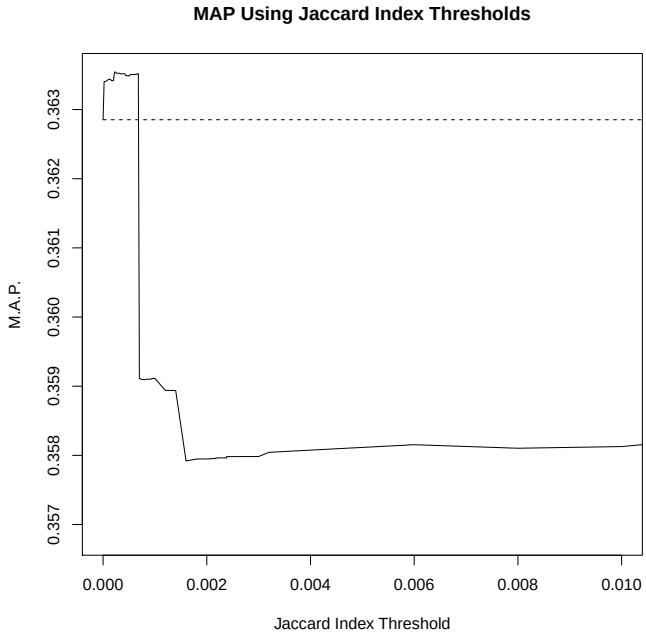

Figure 4.4: Parameter search for a Jaccard Index threshold

the greatest improvement over baseline performance as seen in both Table 4.2 and Figure 4.6. We choose this method for further experiments due to the wide margin between PMI and the other measures.

Performance with different PMI thresholds is not smooth. Figure 4.6 shows several local minima. Better optimisation methods would help with this. KL Divergence also shows a rough surface and several local minima in Figure 4.5.

PMI also had the added benefit of having a stable parameter with a wide range around it. Cosine Similarity had a parameter that is sensitive to small changes as in Figure 4.3 (Note the scale). The performance increase was also negligible compared to the other measures.

KL Divergence and Jaccard Index had only modest increases in MAP, as shown in Figure 4.5 and Figure 4.4.

Based on the findings on the validation set, we tested PMI with a threshold of 1.43 on the test set. Results for all stemmers using this value are shown in Table 4.3. Every stemmer improved, even the S Stripper which has very little conflation and the stemmers that harmed performance.

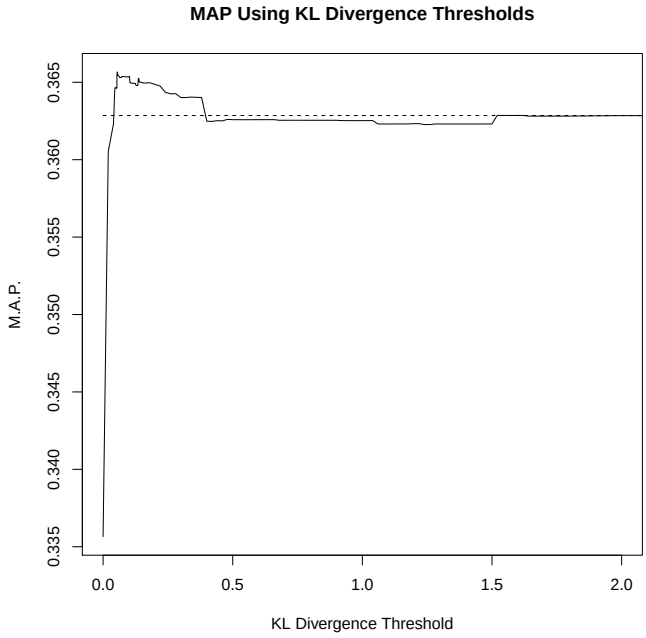

Figure 4.5: Parameter search for a KL Divergence threshold

Conclusion

PMI was able to improve the conflation decisions of a stemmer. The increase was only significant for the Lovins stemmer, but with the small room for improvement and the size of the query set, this is not surprising. We were able to use the parameter found on other stemmers.

4.4.2 Weighting Term Frequencies

For weighting term frequencies we use a sigmoid function to weight term frequencies with their similarity to the original query term. This has two parameters and so involves performing the parameter search over a grid of values.

Results

The results are presented in Figure 4.7, where it can be seen that performance increases around a values of 0.6 to 0.7 and b values between 7 and 12. No pair of values was found for nPMI that would improve the Porter Stemmer. Initially we

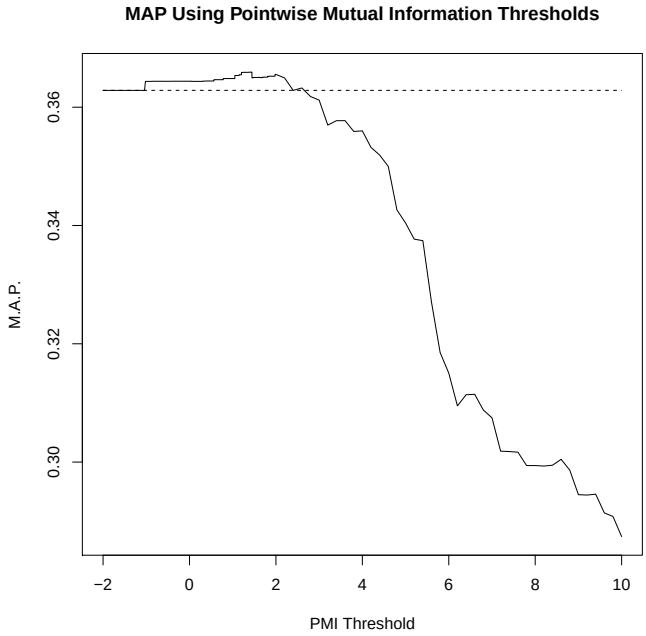

Figure 4.6: Parameter search for a Pointwise Mutual Information threshold

saw that the best values for b were on the edge of the grid at 10, and extended it to 15. As performance stopped increasing at this point, we stopped the search.

Conclusion

The experiment shows that weighting term frequencies fails for the parameters we examined. This failure was unexpected. Since the sigmoid function can represent a threshold we should have done just as well as using a threshold. However to represent the threshold the b parameter must reach infinity. We may have missed values which lead to improvement. We were right to stop searching as there is no evidence that a good parameter is likely to be found in a reasonable time, the performance in Figure 4.7 shows a decreasing trend as b increases.

We might further improve this technique by considering the size of the text window that terms are found in. Terms that occur in different parts of the same document may not be as similar as terms that occur close together in that document. So co-occurrence of terms might be restricted to those that occur together in some window.

Table 4.2: Similarity Measure Thresholds.

	Graph	Parameter found	MAP on training set
Baseline	N/A	N/A	0.3628
EMIM	Figure 4.2	N/A	0.3628
Cosine Similarity	Figure 4.3	6e-13	0.3629
Jaccard Index	Figure 4.4	0.00023	0.3635
KL Divergence	Figure 4.5	2.5	0.3647
PMI	Figure 4.6	1.43	0.3659

Table 4.3: Performance using PMI thresholding on stemmers on the test set. Bold entries have $p < 0.05$.

Stemmer found	original MAP	MAP with PMI
Lovins	0.2969	**0.3239**
Paice/Husk	0.3179	0.3314
No stemming	0.3512	N/A
Porter	0.3806	0.3854
S Stripper	0.3810	0.3824
Ours	0.3858	0.3876

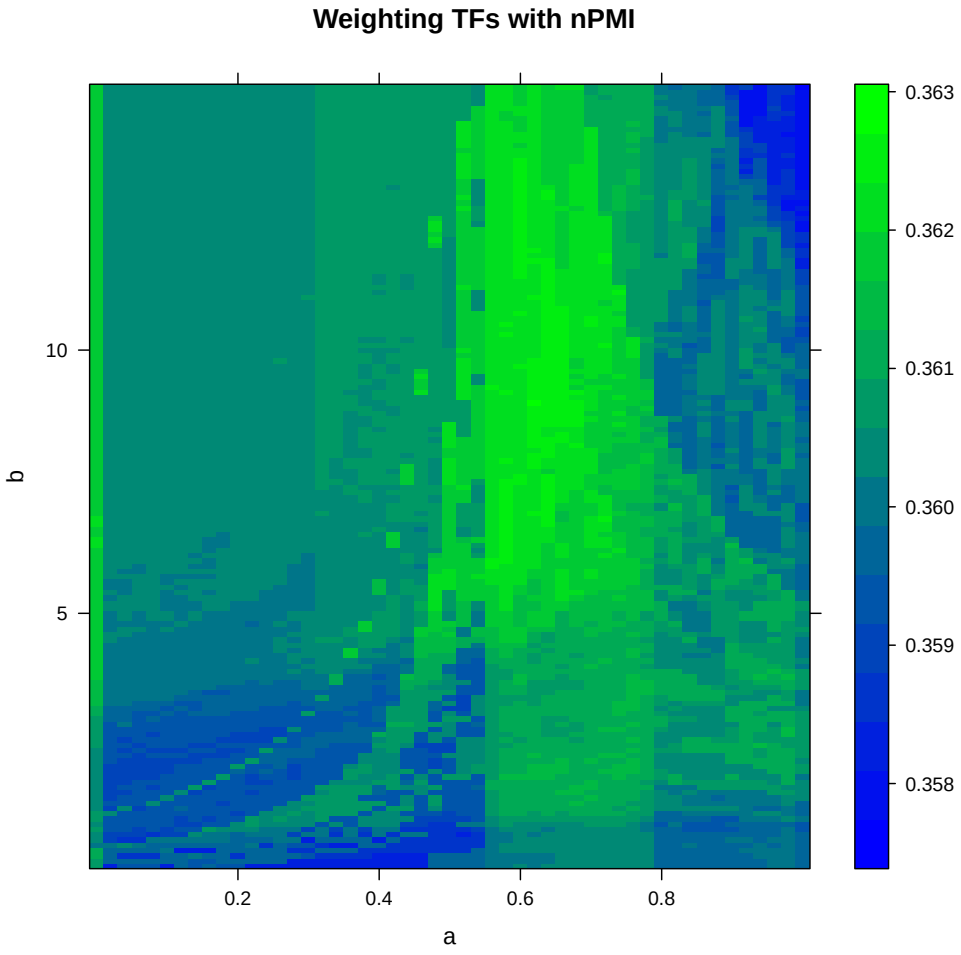

Figure 4.7: MAP over different sigma parameters for weighting TFs with nPMI. No value is better than the performance without weighting.

Chapter 5

When Should We Stem?

Stemming may improve search performance on average, but it can hurt individual queries. We might see larger performance gains if we stem only the queries that improve. In this chapter we examine if there are features of queries or collections that determine the performance of stemming.

Query Performance Predictors (QPP) seek to predict how well queries will perform. We seek to use the principles behind QPP measures to predict whether to use stemming or not. Since we are not predicting the performance of the search engine this problem is new. In this chapter, we will attempt to learn QPP for stemming.

5.1 Current QPP measures

Before creating a formula to predict query performance we examine current measures to see how they perform at the task. This will be the basis of our representation. The features taken from the query, collection or returned documents will be used to form the set of terminals for Genetic Programming.

Only so much information is available before retrieval. As a general rule the more a formula is abstracted from the retrieval process the worse its predictions. The measures that ignore the returned documents will give the same prediction of query performance for any search engine. This limits the predictions to being an average over all search engines. These measures are essentially measuring query difficulty, and predict that whatever search method you choose will perform better on the easier queries. Of course, queries are seen as easy precisely when search engines perform well on them. Some measures take a further step and ignore information in the collection and use only information in the query itself. The predictions will

be the same for a query across all document collections. These should perform even worse.

Some assume that hard queries are ambiguous (He and Ounis, 2004). If a query is ambiguous then the retrieval will suffer since we must cater to all possible meanings of the query. Others attempt to discover how good a query is by examining the returned documents (Cronen-Townsend, Zhou, and Croft, 2002b). If the retrieved documents are similar to the whole collection then the query has insufficient selective power to pick relevant documents.

5.1.1 Measuring Specificity

The following measure how well a query will pick documents. Inverse Document Frequency (IDF) does this for individual terms in a query. A term has high specificity if it appears in few documents. A query with terms with low specificity will pick out large numbers of documents. A query without any terms of high specificity will give a search engine a harder time ranking them. Query Length (QL) and Average Query Length ($AvQL$) are based solely on the query, these are interesting in that they will work without a particular collection but should be bad at predicting performance.

Query Length (QL) (Zhai and Lafferty, 2004) is the number of terms in a query (Q). If a query requires a large number of words to specify it, it is trying to be very specific. On the other hand noise from each of the terms is likely to accumulate. He and Ounis (2004) showed that QL has little correlation with difficulty and we find the same in Table 5.1. We made it available to learning despite this as it is often used in more complex formulas. This is measured after removing stop words.

Let Q be the set of query terms.

$$QL = |Q|$$

Average Query Length ($AvQL$) is the average length of words in a query. It considers queries with longer words to be better, since longer words are less common. Being rarer words, they will select a smaller set of documents. IDF is a more direct measure of this. The measures we examine that ignore collection information have low correlation with performance as in Table 5.1.

Let Q be the set of query terms and $strlen(q)$ be the length of term q in characters.

Table 5.1: Kendall's Tau B Correlation between QPP Measures, AP and δAP when stemming. Bold entries have p < 0.05. Figures are from the training set.

Measure	**AP Tau**	**δAP Tau**
QL	-0.0020	-0.0386
AvSCQ	**-0.1700**	**-0.0773**
AvSCS	**0.1448**	**0.0877**
AvQL	-0.0237	0.0249
AvIDF	**0.1636**	0.0474
MaxIDF	**0.1517**	0.0368
AvICTF	**0.1481**	**0.0933**
DocsReturned	**-0.1027**	-0.0097
TopDocs	**-0.1303**	-0.0269
Top BM25 Score	**0.2123**	0.0175
δ DocsReturned	0.0547	0.0567
δ TopDocs	0.0097	0.0343
δ Top BM25 Score	**-0.0875**	0.0487
QueryScope	**0.1027**	0.0097
StdDevIDF	**0.1458**	0.0443

$$AvQL = \frac{1}{|Q|} \sum_{q \in Q} strlen(q)$$

Maximum Inverse Document Frequency ($MaxIDF$) is the largest IDF that any word in the query has. IDF is the log-inverse of the frequency of documents in the collection that have the term (q). The term with the largest IDF will probably be the one with the most impact on ranking. A term with a high IDF will pick out a small number of documents. A query without some high IDF terms is unlikely to select particular documents.

Let $d(q)$ be the set of documents that contain term q, and Q be the set of query terms.

$$IDF(q) = log \frac{|D|}{|d(q)|}$$
$$MaxIDF = \max_{q \in Q} IDF(q)$$

Average Inverse Document Frequency ($AvIDF$) is the mean of the IDF scores of the terms in the query. This relies on the same principles as $MaxIDF$, but takes into account that terms other than the one with highest IDF will still have an impact on ranking. However terms with low IDF will have an effect on $AvIDF$, but almost no effect on ranking. $StdDevIDF$ is the standard deviation of the IDF scores for the terms in a query.

$$AvIDF = \frac{1}{|Q|} \sum_{q \in Q} IDF(q)$$

Average Inverse Collection Term Frequency ($AvICTF$) (He and Ounis, 2004) is the mean average of the inverse frequency of query terms in the entire collection. Where IDF counts the number of documents a term occurs in, $ICTF$ counts the individual occurrences within all documents. This accounts for multiple occurrences of terms within one document

Let $TF(q, d)$ give the frequency of term q within document d, and D be the set of documents returned by the query.

$$CTF(q) = \sum_{d \in D} TF(q, d)$$

$$ICTF(q) = log \frac{|D|}{|CTF_q|}$$

$$AvICTF = \frac{1}{|Q|} \prod_{q \in Q} ICTF(q)$$

Query Scope is the negative log of the proportion of the documents returned. Queries that return small parts of the collections are assumed to be better at picking out relevant documents. However very common words, which have little effect on ranking, will alter the Query Scope drastically. For larger queries the quality of this measure decreases as the number of documents containing at least one term will tend towards the entire collection. For single word queries it reduces to the negative IDF. This was introduced by He and Ounis (2004).

Let D be the set of returned documents and C be the set of documents in the collection.

$$QueryScope = -\log \frac{|D|}{|C|}$$

5.1.2 Difference between query and collection

Simplified Clarity Score ($AvSCS$) (He and Ounis, 2004) measures how different the language models of the query and document collection are. The formula below is simplified using the assumption that terms in the query will only occur once. KL Divergence is used to measure the actual difference.

$$AvSCS = \frac{1}{|Q|} \sum_{q \in Q} \log_2 \frac{ICTF(q)}{|Q|})$$

Average Collection Query Score ($AvSCQ$) determines the similarity of the query and collection using the vector space model. Treating each term as a dimension, it treats the query, documents and the entire collection as vectors. Distance along each dimension is determined by the occurrence of terms in the query, document or collection. It then measures similarity of the vectors. This was put forward and evaluated by Zhao, Scholer, and Tsegay (2008) with further methods based on $AvSCQ$. Due to our use of summary statistics for sets of numbers, this formula is not learnable by our Genetic Programming.

Let $DF(q)$ be the document frequency of a term q.

$$AvSCQ = \frac{1}{|Q|} \sum_{q \in Q} (1 + \ln CTF(q)) * \ln(1 + \frac{|D|}{DF(q)})$$

5.1.3 Differences between query terms

Average Expected Mutual Information Measure ($AvEMIM$) is the mean average EMIM between each pair of query terms. This measures how similar each pair of terms is. A query made of terms that occur together regularly will rank those documents highly. Queries with high $AvEMIM$ have terms that occur in the same documents. Documents with many query terms are usually the top ranked documents. Queries with low $AvEMIM$ will have documents with different query terms compete for the top rankings.

Let $P(q_1, \ldots, q_k)$ be the estimated probability of $q_1, \ldots, q_k$ occuring within the same document. Let $i < j$, so we choose without replacement.

$$AvEMIM = \frac{1}{\binom{|Q|}{2}} \sum_{q_i, q_j \in Q} EMIM(q_i, q_j)$$

Average Pointwise Mutual Information ($AvPMI$) is the average PMI between each pair of query terms. This examines the probability of the two terms occurring in

the same document, using the collection as the sample. This measures how similar each pair of query terms are. This was examined by Hauff (2010). A query with very similar terms provides more information about the same documents, which allows for better ranking.

$$AvPMI = \frac{1}{\binom{|Q|}{2}} \sum_{q_i, q_j \in Q} log_2 \left(\frac{P(q_i, q_j)}{P(q_i)P(q_j)} \right)$$

5.1.4 Ranking Function as QPP

When we rank documents for the user, we estimate how well each document will fulfill the information need. This estimate could help prediction of how well the information need will be met by the ranked list of documents.

Okapi BM255 (Robertson *et al.*, 1996) is a ranking function that itself correlates well with Average Precision scores. Initial work using just the score of the top ranked document showed good correlation with AP compared with other measures in Table 5.1. When we examine the change between stemming and no stemming in the top scores it also correlated well with change in AP, as shown in Table 5.1.

A further improvement upon the raw score is the adjusted formula below. This normalisation is pointless when using $BM25$ to rank. When comparing scores between queries this is necessary to adjust scores. Otherwise longer queries will have higher scores, even if they merely contain more copies of the same terms. This score penalises queries with terms that contribute little, so removing terms with low IDF is necessary.

$$BM25_{adj.}(Q, doc) = \frac{1}{|Q|} BM25(Q, doc)$$

We look at both the $TopBM25$ value as a QPP measure, and the change in the top $BM25_{adj.}$ value for stemming and without stemming, which is $\delta TopBM25Score$.

5.1.5 Other QPP measures

The measures examined thus far are simple and easy to implement. They also take little time to calculate. There are more that are more complex and slower to calculate. These either examine the entire collection or subsets of documents in ways that would be difficult to precalculate. Attempting to represent these would complicate

learning. We try the simple approach and make no attempt to represent them. Most depend on performing the query, and examining the resulting set of documents.

Clarity (Cronen-Townsend *et al.*, 2002b) is the KL divergence between the language model produced by the results and the collection as a whole. This measures how well the query has selected documents out of the collection. The language model of the results uses the top n documents of the results for a query. Improved clarity is an attempt at fixing the decision of how many documents to use for the results language model. It also improves the predictions.

In addition we considered the number of documents returned by answering the query ($DocsReturned$) as well as the number of documents with the greatest number of matched terms in the query ($TopDocs$) as QPP measures. These were examined because we wanted to use them as variables during learning. Both can only increase during stemming. $\delta TopDocs$ and $\delta DocsReturned$ are versions of $TopDocs$ and $DocsReturned$ respectively that are the difference between the stemmed and unstemmed scores.

5.2 Our Fitness Function

The literature uses Kendall's τ to determine how well a QPP predicts (He and Ounis, 2004)(Zhao *et al.*, 2008). If a QPP formula correlates well with the AP of queries, then it is a good predictor of performance. We deviate from this because we are more interested in the effects of using a formula to decide when to stem. This allows us to measure the impact of a QPP formula upon the MAP of a query set directly.

The particular rank is less important than whether the rank is high or low. There is a single point beyond which stemming is not used. For example if the top items according to one score are the top items for another the particular ordering does not matter as long as the top scores are similar. Those where stemming will improve searching and those where it will not.

We measure how well the predictor would do at selecting between stemming and no stemming for actual queries. This is the performance it would have if we use the predictor to decide when to stem. We find the best value to use as a threshold for stemming. We use the same threshold values on the validation and test queries that were found on the training queries.

For a measure like the δ top score, a threshold of zero would be equivalent to choosing whichever method has the highest score. We might find that we can get

away with stemming more often than when it has the highest score, and so our threshold should be lower.

Our fitness value is the MAP score of the queries with stemming applied selectively. So there is a maximum fitness where we make perfect decisions each time. There are also two baselines, the lower is to never stem, the other is to always stem. A bad predictor can do worse than never stemming by picking stemming when it hurts performance. The baseline we want to beat is always stemming. A QPP formula has succeeded if it gives better performance than always stemming.

For the experiments we use the Porter stemmer, as this has a more variable performance than the stemmer we created. This can be seen by comparing Figure 5.1 and Figure 5.2. This also allows us to use the same query sets we learnt our stemmer with.

Table 5.2: Genetic Programming Parameters

Population Size	1000
Function Set	+ - / * log
Terminals	Variables listed in 5.3.1, and Constants between [0.0, 1.0]
Generation Method	Ramped Half-and-half
Crossover Rate	0.8
Mutation Rate	0.2
Max Depth	20
Min Depth	4
Max Generations	1000

5.3 Representing QPP Formulae

The representation used initially for the learning of QPP measures is an abstract syntax tree. This is typical for Symbolic Regression performed by Genetic Programming (Koza, 1992). The set of functions used are those found in other QPP measures. They are typical for this kind of learning.

The choice of function set is a collection of functions used in the QPP measures above. Variable choice is also important, and consists of pre-retrieval statistics and some simple post-retrieval statistics.

To avoid the problem of learning formulae with scoped operations, such as taking the sum of a set of numbers, we use summary statistics of the sets examined in QPP measures. For example, where other measures might sum the lengths of query terms, we have the average of the lengths. So instead of allowing summation and similar operators into the function set, we have the results of common operators as terminals. This means we cannot represent complex formulas with suboperations that occur in a scoped operator. Rather we provide summary statistics of variables that are sets or ranked lists. For sets of values these are the maximum, minimum, mean and standard deviation of the set. These allow us to examine the values for the longest query term, or the average length of query terms. For ranked lists, like the top scores of the ranked document list, we provide the top score, and averages of the five top scores and the ten top scores.

5.3.1 Terminals

Features used in the QPP measures described above are statistics drawn from the query, collection, or the results of performing the query. A few provided are comparisons between the two sets of results with and without stemming. These are all easily gathered before any learning takes place, though many would change if the stemmer, search engine, query set or collection were changed.

When calculating some of the QPP measures often a scoped operations like summation is required. These deal with sets of numbers. It is hard to reconcile such operations with Genetic Programming. One of the features of Genetic Programming that make crossover and mutation such nice operations is the closure of functions. If we have function closure, each function only needs to refer to the results of its children. For scoped operations to be useful they would need to introduce a new variable terminal that would be used in the children of the operation. This is harder to implement than is worth doing. As such we instead use summary statistics of those features that are not scalar values.

Floating point constants are also used as terminals, providing random values between 0 and 1, which are fixed on generation. Testing has shown the ability of intron branches to use these to create much larger values if needed. These values allow branches to be weighted during the learning process. Few of the referenced QPP formulas include constants other than 1.

Collection Size

Collection size is constant for each query set. QPP measures use this constant a lot, and we provide it to avoid the need to learn it. It is generally used to convert some number of documents into a proportion of all documents.

Query Length

Query length is the number of terms in the query. Like the collection size it is often used to scale numbers into an average over each query term, as in AvQL. It varies with the query, and has been used as a QPP measure by itself, as in Table 5.1.

Term Length

The term lengths are the lengths of terms in the queries in characters. Average term length is examined previously as a QPP measure. We provide some summary statistics, rather than using the set of numbers directly. So we have the minimum, maximum, mean and standard deviation of the term lengths as scalar terminals.

Returned Documents

The number of returned documents can approximate recall. Unfortunately it is suceptible to very common terms. Any query including 'the' is likely to return all documents in the collection. The queries are stopped; we remove very common words that occur in more than half the documents. The number of returned documents for stemming is always at least the number for not stemming.

A delta version of this is also used. This is the difference between returned documents with stemming and the returned documents without stemming. Stemming can only increase the number of documents returned, so this is always positive.

Maximum Matched Terms

This is the maximum number of matched terms found in a single document. This may be a good measure of how well the query is doing. A query with few matched terms together in a document may need stemming more to find other versions of those terms. This is greater for stemming than not stemming.

Equivalence Class Size

The equivalence class size is a vector of the size of the classes created by the stemmer for each term. These classes are the terms that are conflated together. So the size of the equivalence class for a term, is the number of terms that have been conflated together with it (including the term itself.)

Collection Term Frequency

Another vector measure, collection term frequency, is the number of times the term occurs in the collection. This counts each time the term occurs within the same document. We give the minimum, maximum, mean, and standard deviation of the collection term frequencies of query terms as terminals.

Document Frequency

Document frequency is much like collection term frequency, except it only counts occurrence in a document once. So it is the number of documents a term occurs in. As a vector, we give the four summary statistics as the other vectors.

BM25 Score

Unlike the other vector measures, the scores are ranked. To better summarise this, we provide the top score and the means of the top five and top ten scores. In addition we use the standard deviation of the top ten scores.

Delta versions of these are also used. For each summary statistic we create a delta version by taking the difference of the values for stemming and without stemming.

Term Similarity Measures

We use the set of term similarity measures to see how similar the terms in a query are. Since we use co-occurrence in documents to do this, this is a reasonable way of determining how many documents are likely to be highly scored.

Cosine similarity and the other similarity measures are vectors that compare the similarity between each pair of terms. For queries of a single term this is defined as one; terms are considered perfectly similar to themselves.

Again, the minimum maximum, mean and standard deviation are used as scalar summaries.

Table 5.3: δAP when measures are used as a deciding parameter for choosing when to stem on the test set.

Measure	δ**AP**
QL	0.027438
AvSCQ	0.026087
AvSCS	0.027929
AvQL	0.024689
AvIDF	0.026429
MaxIDF	0.026429
AvICTF	0.030402
DocsReturned	0.009310
TopDocs	0.023671
Top BM25 Score	0.027519
δ DocsReturned	0.0
δ TopDocs	0.0
δ Top BM25 Score	0.033956
QueryScope	0.025595
StdDevIDF	0.026429

5.3.2 Function Set

We use all the major functions that are in the set of QPP measures previously examined. This consists of the basic binary operators of addition, subtraction, multiplication and division. Division is protected so that division by zero returns zero. Also included are unary negation, and a base 10 logarithm. Logarithms are often used in IR formulas to reduce large numbers to their order of magnitude. An example of this is the use of the logarithm in calculating IDF where the logarithm prevents overflow.

5.4 Stemming individual queries

When using Rank Effectiveness to rate queries rather than Average Precision, we find that those queries which become worse with stemming improve as in Figure 5.1 and Figure 5.2. The bars which go below zero are much shorter using Rank Effectiveness. So the stemmer is being penalised for finding documents of unknown

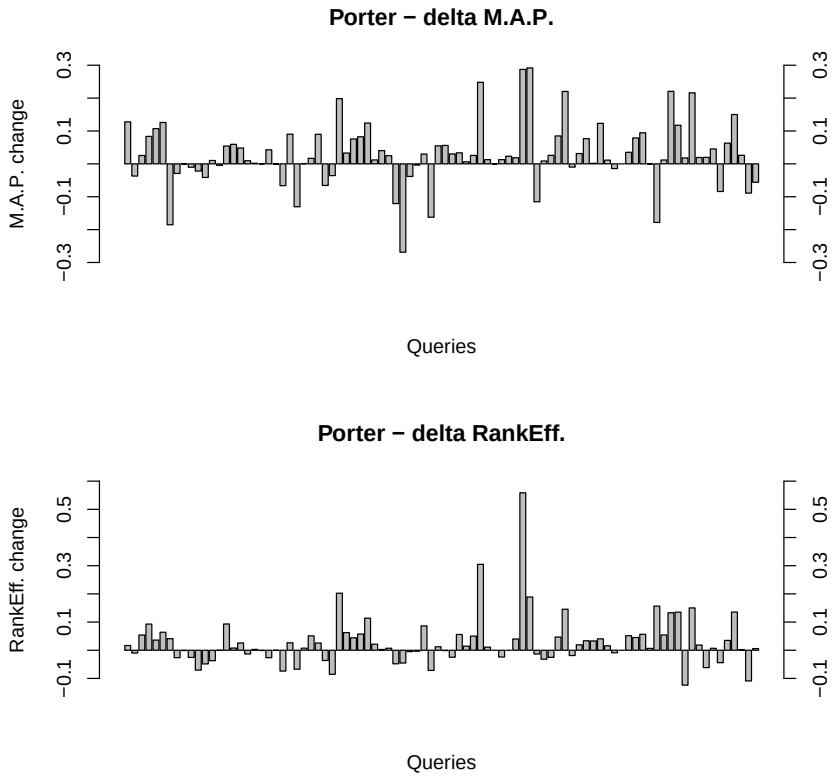

Figure 5.1: A comparison on the differences using M.A.P. to rate the per query improvements made by the Porter stemmer, and using RankEff.

relevance.

The difference between the measures is that Rank Effectiveness ignores documents which are not rated. In most cases the documents which are being ranked above relevant ones will not be rated. Stemming cannot remove documents from the retrieved list, but their contribution to Rank Effectiveness and Average Precision will decrease with their rank.

Determining the true range of values is difficult as calculating Average Precision involves knowing the total number of relevant documents.

This suggests that learning using Rank Effectiveness as a fitness function for learning Stemmers rather than M.A.P. would produce stronger stemmers.

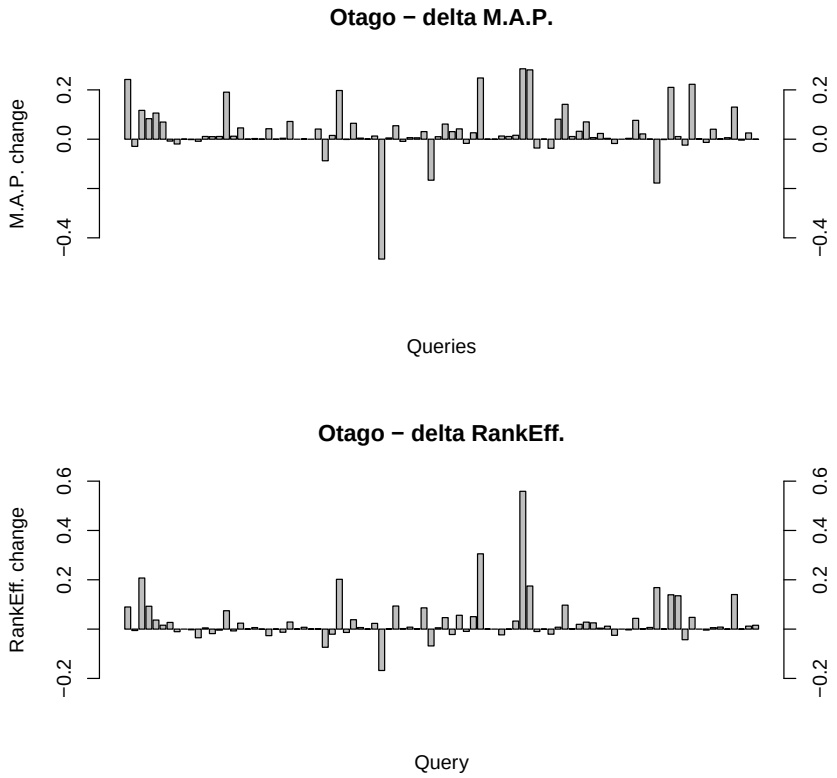

Figure 5.2: A comparison of the differences using M.A.P. to rate the per query improvements made by the Otago stemmer, and using RankEff.

5.5 Experiments

The first experiment of examining the effect of document sizes on stemming performance was performed using queries 150-200 on the Wall Street Journal collection. The later experiments used both the 2008 and 2009 versions of the Wikipedia Collection using the 2006-2010 queries from INEX.

5.5.1 Corpus Document size

We investigate if the lengths of documents in the collection determine if we should stem. This has been done previously by Hull (1996), but they looked at smaller collections that differed on more than document size and had different queries.

We created three versions of the Wall Street Journal collection. One consisted of the title tags of the documents, the second was the first paragraphs of the collection,

and the final version was the original collection. These give simple and natural cut-off points simulating three collections with radically different document sizes.

Results

Changes in performance may be seen in Figure 5.3. Collections with shorter documents have consistently greater improvement by stemmers. However the total performance is much greater when the documents are longer. All stemmers still improve the original collection.

Conclusion

Stemming is much more important for shorter documents. Longer documents are more likely to match different stems. Not all collection sizes are increasing, Twitter is an example of a current service with very short documents.

Extending documents is not as easy as truncating them, and so a different collection that provides longer documents would be required to extend this investigation further. Extending the length of documents may be done from additional sources. This shows the importance of stemming within focused retrieval where search attempts to find relevant parts of documents.

5.5.2 Existing Measures

We first compare the ability of existing measures to predict Average Precision of queries. This is measured by the τ between the value of the measure for a query, and the Average Precision of that query.

Comparing the predictive power of these measure to predict Average Precision provides some idea of how well the basic measures do. Examining the predictive power of the same measures to predict the change in Average Precision is a more direct measure of how well they should predict when to stem.

Results

Correlation values are found in Table 5.1. Some measures are able to show significant correlation with Average Precision of queries. When extended to predicting the difference in Average Precision when stemming, every correlation is weaker with many failing to be significant.

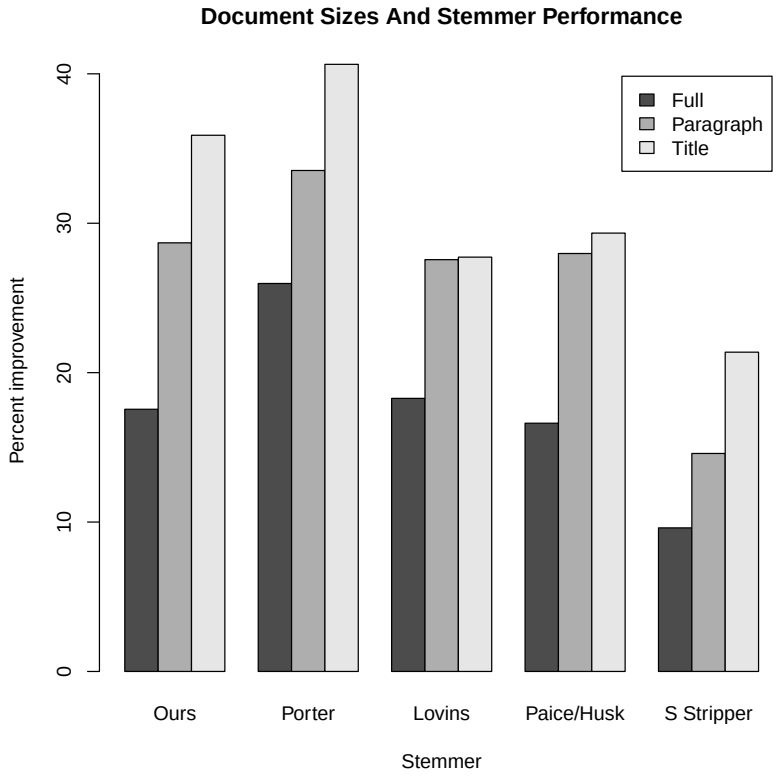

Figure 5.3: A comparison of Percentage Improvement made by stemmers for versions of the same collection. One collection was shortened to the first paragraph, one was shortened to document titles.

Conclusion

We conclude that while these measures might give a weak indication of query performance, they could not be used to predict it. They are even worse at selecting which queries to stem. We cannot use any of these measures to refine our stemming per query. This means creating our own formula designed to predict when stemming will be beneficial would be worthwhile.

5.5.3 Learning parameters

Before we can use Genetic Programming to learn formulas effectively we needed to find effective learning parameters. First we examined different crossover and mutation rates. Then we examined different population sizes. Results are based on the average of twenty runs per parameter setting. This is higher than the experiments in Chapter 3, due to the speed of the task.

We examined crossover at rates of 60, 70, 80 and 90 percent along with mutation rates of 0, 10, 20, and 30 percent. These were chosen as they are similar to the rates used elsewhere. Populations sizes of 400, 600, 800, and 1000 were chosen to be reasonable based on the time each run took.

Table 5.4: Change in Mean Average Precision over Crossover and Mutation Rates.

		Mutation Rate			
		0%	10%	20%	30%
	60%	0.03448	0.03341	0.03493	0.03534
Crossover	70%	0.03267	0.03493	0.03526	0.03520
Rate	80%	0.03387	0.03402	**0.03548**	N/A
	90%	0.03397	0.03523	N/A	N/A

Results

Learning curves for combinations of crossover rate and mutation rate are shown in Figure 5.4. The final fitness values are in Table 5.4. The best combination of crossover and mutation is a crossover rate of 80% and mutation rate of 20%. These differ from the ones found in Chapter 3.

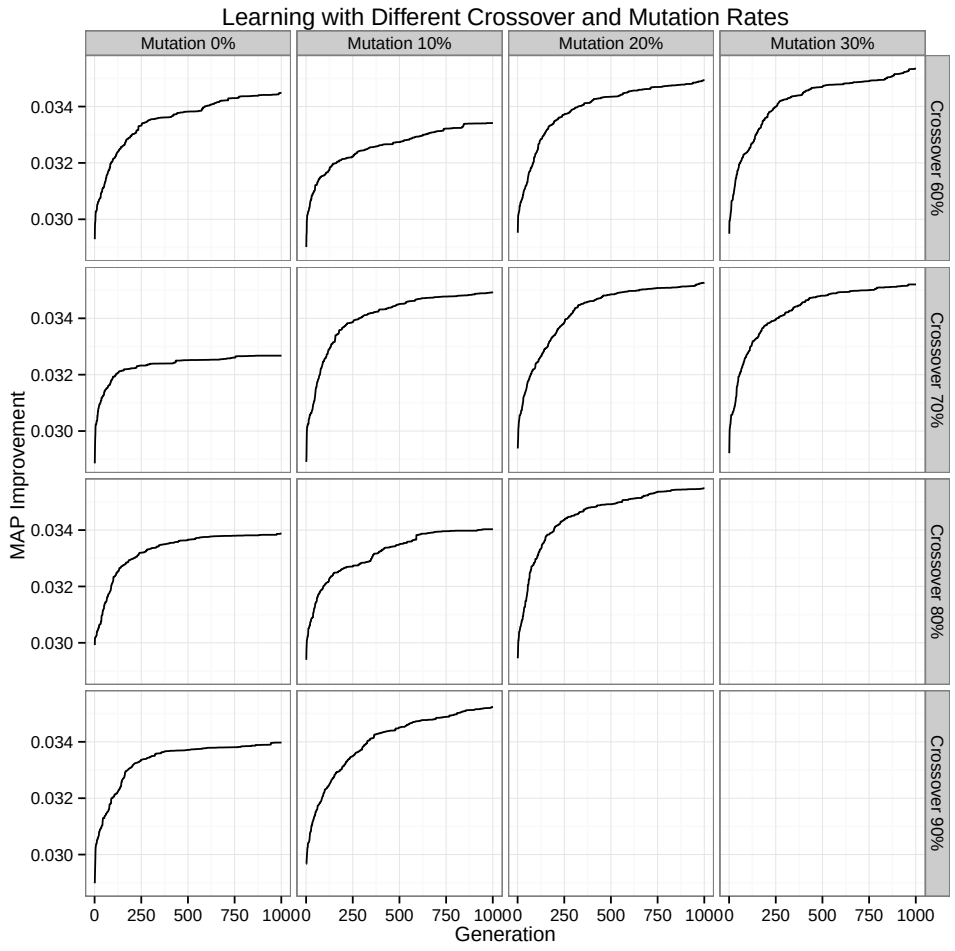

Figure 5.4: Average fitness over generations for different rates of crossover and mutation.

Curves are given for different population sizes in Figure 5.5 and Figure 5.6. Treating individuals evaluated as a more accurate approximation of time, we focus on the population size that performs better for the individuals examined. In this case the largest population size performs the best.

5.5.4 Final Learning

We performed 115 runs using the found parameters. The best individuals were evaluated on the validation set, and the best of those was evaluated on the test set. These sets are the same as used in previous chapters.

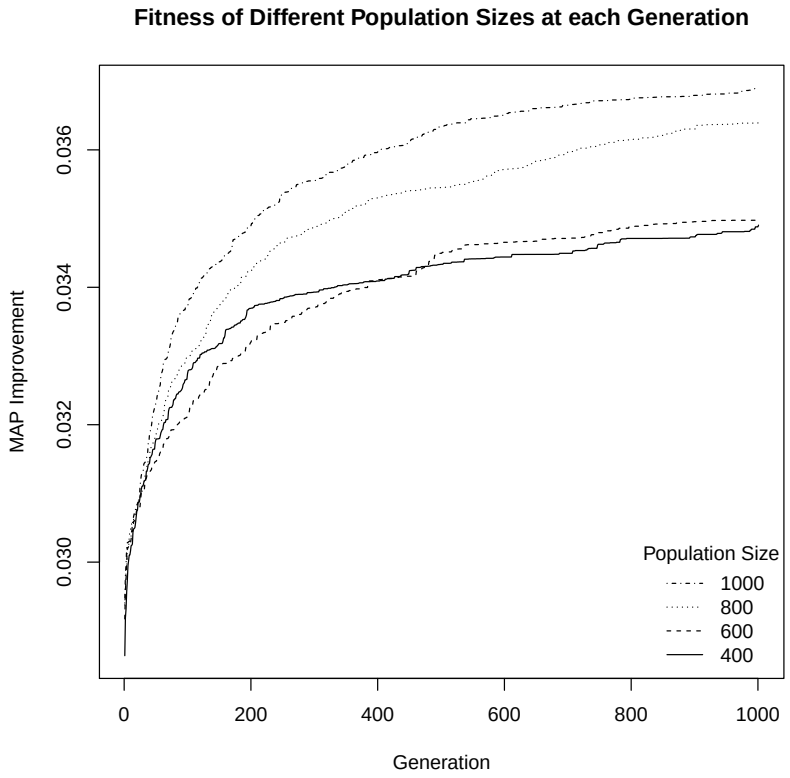

Figure 5.5: Average fitness over generations for different population sizes.

Results

Performance of formulae on the validation and training sets is shown in Figure 5.7. The final formula is the point at the top. This formula fails to beat the baseline of always stemming on the test set.

The resultant formula is this:

$$\frac{\text{min KL divergence}(-34 - \delta\text{top 10 scores})}{\text{top 5 scores} \times \text{mean equivalence class size}}$$

We used a greedy form of simplification to arrive at this formula. Any subtree of the formula that was constant on the training data was replaced by that constant. Any node that could be replaced by a child node without degrading performance was swapped with whichever child performed best in its place.

Simplification improved performance on the test set by 0.002 MAP. Performance on the test set did not pass the baseline performance of always stemming. Our final QPP formula has an extremely low correlation of 0.095 with the δAP of the

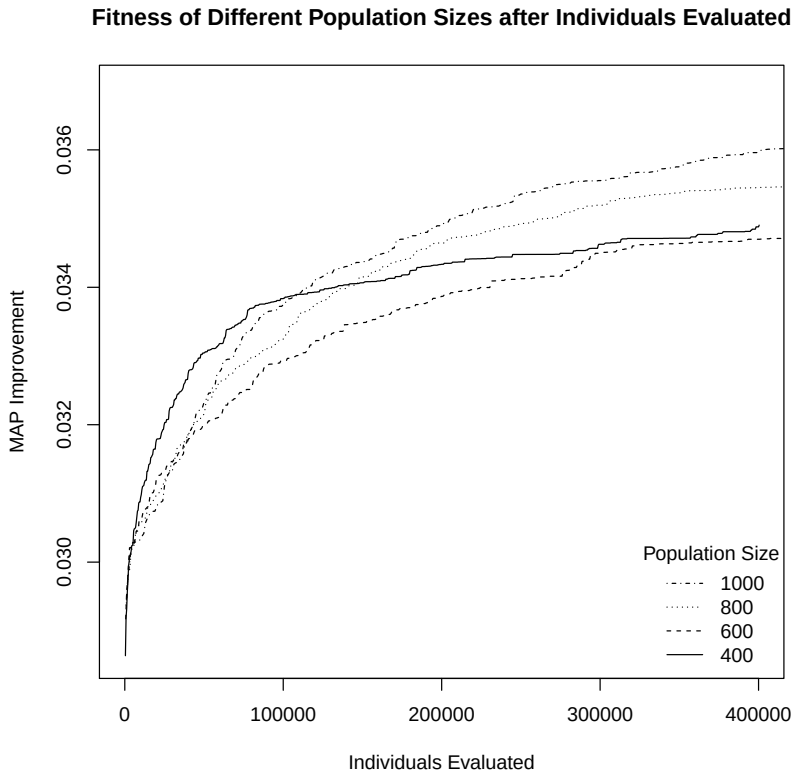

Figure 5.6: Average fitness over individuals examined for different population sizes.

test queries, which is insignificant. Performance at deciding when to stem is poor. The result of using this formula is an MAP increase of 0.0251, which is less than the 0.0294 increase from always stemming.

The best of the individual terminals, $\delta top5BM25score$ gave an increase of 0.0352 MAP. That is, when the top five normalised BM25 scores for the stemmed results was more than -2.72 that of the top five scores for the non-stemmed results the stemmed results should be used. This increase was not significant at $p < 0.05$. The greatest increase possible was 0.0482 MAP if it could predict perfectly.

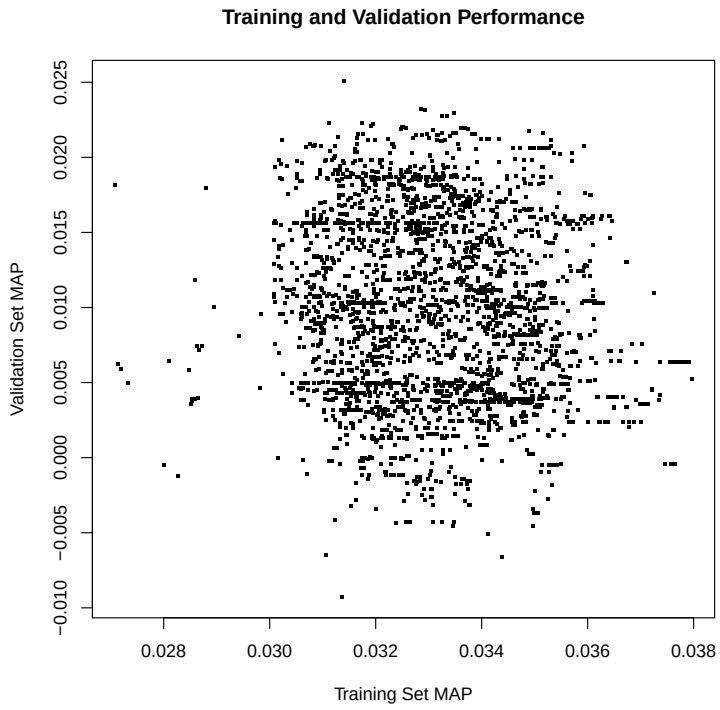

Figure 5.7: MAP for each QPP formula on training and validation query sets.

5.6 Conclusion

Stemming is more important for collections made up of smaller documents. Since longer documents are more likely to contain multiple versions of the words they are concerned with, this is unsurprising.

While comparing QPP measures using correlation measures fits well with our results, more direct evaluation of the measures is required for them to be useful. We believe focusing on measuring individual prediction tasks is better than striving for general predictors.

That learning failed to produce a successful QPP formula, while a terminal succeeded in predicting at least enough to beat our best formula may indicate a difficulty in generalising with formulas found by Genetic Programming. This also shows the importance of investigating the ability of the terminals to perform the task. Since Genetic Programming included a minimum tree size, learning these is harder than it should be.

Using a ranking function seems the best way of indicating performance. The

task of predicting how results will be relevant is obviously related to ranking documents. In a way, ranking documents to answer a query is predicting performance. The scores that are used to rank documents are predictions of their relevance already. We do not know if just any ranking function would be effective at predicting performance. Remember that the ranking function we used to predict was the same used to rank our results. We cannot assume other ranking functions would predict as well as Okapi BM25 did.

Ranking functions are already attempts to predict the relevance of individuals documents for a query. The ranking function scores attempt to rank functions in order of how likely they are to be relevant. All that is required to predict query performance is a summary of the results' scores. This is what $\delta top5scores$ was.

If a QPP metric could predict when results are relevant then it should be usable as a ranking function. Insights from QPP measures could be used to understand ranking. Improvements in predicting performance imply improvements in ranking. So we believe the QPP research project should be a part of ranking function research.

Chapter 6

Conclusions

6.1 Summary

Vocabulary mismatch is a problem for information retrieval. Stemmers are an attempt to solve that problem, but have been handwritten. We sought to create a method for using machine learning to create a stemmer optimised for information retrieval performance. Our representation is symbolic and allows us to examine why the rules were found to be successful, and we found interactions between the rules being learnt. These were not just individual rules, but were rules that worked together to conflate words. The final product was a weak stemmer optimised purely for retrieval performance. That stemmer was able to generalise on unseen queries, and performed comparably to the current state of the art.

Stemmers ignored the document collection as source of information. This is a valid, valuable source of information regarding the language used in the collection. Given the multiple term similarity measures available we had to find one that would work for the task. Pointwise mutual information was shown to be an effective measure to adjust stemming decisions within a stemmer. Our stemming decisions now utilise information about how language is used in the collection, a source normally ignored.

Measures available to tell us when to use stemming are poor. If there were a predictor that performed better we would be able to use the predictions to improve stemming. Our machine learning failed to give us a good predictor, but pointed us in the direction of using the ranking function as a QPP measure. Ranking functions already predict the relevance of individual documents. Extending this to predict how well results fulfill a query seems to be as easy as summarising the top results

scores.

The products of all three parts of this thesis, learning stemmers, improving stemmers with pointwise mutual information, and predicting when to stem, have the ability to be used together to improve retrieval performance.

6.2 Future Work

The effect of adjusting stemmer decisions will have an effect on what rules should be learnt. As this affects the decisions made by the stemmer, it is likely a much stronger stemmer would be learnt if this was used during learning.

Our method of learning should generalise to similar languages that need suffix rules and to specialised document collections. Specialising a stemmer for a particular collection should provide a greater performance increase than a more general stemmer. We have yet to see how effective our representation is for other languages.

Results of QPP research should be re-examined in the hope of providing insight into ranking functions. The problems of ranking results and predicting the performance on a query appear to be highly related.

Stemmers for IR should be focused to perform well at the tasks we put them to. Machine learning, in particular symbolic learning, has the ability to create task-focused stemmers with an understandable representation.

References

Ahlgren, P. and Grönqvist, L. (2008). Evaluation of retrieval effectiveness with incomplete relevance data: Theoretical and experimental comparison of three measures. *Information Processing & Management*, *44*(1), 212–225.

Anscombe, F. J. (1973). Graphs in Statistical Analysis. *The American Statistician*, *27*(1), 17–21.

Bouma, G. (2009). Normalized (Pointwise) Mutual Information in Collocation Extraction. In *Proc. Biennial GSCL Conference 2009, Meaning: Processing Texts Automatically*, 31–40. Tübingen, Gunter Narr Verlag.

Croft, W. B. and Xu, J. (1995). Corpus-Specific Stemming using Word Form Cooccurrence. In *In Fourth Annual Symposium on Document Analysis and Information Retrieval*, 147–159.

Cronen-Townsend, S., Zhou, Y., and Croft, W. B. (2002). Predicting query performance. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 299–306. ACM.

Cronen-Townsend, S., Zhou, Y., and Croft, W. B. (2002). Predicting query performance. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, New York, NY, USA, 299–306. ACM.

Denoyer, L. and Gallinari, P. (2006). The Wikipedia XML Corpus. *SIGIR Forum*.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Weslay.

Gordon, M. (1988). Probabilistic and genetic algorithms in document retrieval. *Commun. ACM*, *31*, 1208–1218.

Grivolla, J., Jourlin, P., and Mori, R. D. (2005). Automatic Classification of Queries by Expected Retrieval Performance. *ACM SIGIR '05*.

Harman, D. (1991). How effective is suffixing? *Journal of the American Society for Information Science*, *42*(1), 7–15.

Hauff, C. (2010). Predicting the Effectiveness of Queries and Retrieval Systems.

Hauff, C. and Azzopardi, L. (2009). When is query performance prediction effective? In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 829–830. ACM.

He, B. and Ounis, I. (2004). Inferring query performance using pre-retrieval predictors. In *In Proc. Symposium on String Processing and Information Retrieval*, 43–54. Springer Verlag.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press.

Hull, D. A. (1996). Stemming Algorithms - A Case Study for Detailed Evaluation. *Journal of the American Society for Information Science*, *47*, 70–84.

Jacobs, J. (1982). Finding words that sound alike. The SOUNDEX algorithm. *Byte 7*, 473–474.

Koza, J. R. (1992). *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press.

Lovins, J. (1968). Development of a Stemming Algorithm. *Mechanical translation and computational linguistics*, *11*, 22–31.

Macdonald, C., He, B., and Ounis, I. (2005). Predicting Query Performance in Intranet Search. *ACM SIGIR '05 Query Prediction Workshop*.

Mayfield, J. and McNamee, P. (2003). Single n-gram stemming. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, New York, NY, USA, 415–416. ACM.

Paice, C. D. (1990). Another stemmer. *SIGIR Forum*, *24*, 56–61.

Paik, J. H., Pal, D., and Parui, S. K. (2011). A novel corpus-based stemming algorithm using co-occurrence statistics. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, SIGIR '11, New York, NY, USA, 863–872. ACM.

Peng, F., Ahmed, N., Li, X., and Lu, Y. (2007). Context sensitive stemming for web search. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, New York, NY, USA, 639–646. ACM.

Philips, L. (1990). Hanging on the Metaphone. *Computer Language*, *7*(12 (December)).

Porter, M. (1980). An Algorithm for Suffix Stripping. *Program*, *14*(3), 130–137.

Porter, M. (2010). Snowball: A language for Stemming Algorithms. `http://snowball.tartarus.org/texts/introduction.html`.

Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M., and Gatford, M. (1996). Okapi at TREC-3. 109–126.

Smucker, M. D., Allan, J., and Carterette, B. (2007). A comparison of statistical significance tests for information retrieval evaluation. In M. J. Silva, A. H. F. Laender, R. A. Baeza-Yates, D. L. McGuinness, B. Olstad, Ø. H. Olsen, and A. O. Falcão (Eds.), *CIKM*, 623–632. ACM.

Spärck Jones, K. (1971). *Automatic Keyword Classification for Information Retrieval*. Archon Books.

Trotman, A. (2004). An artificial intelligence approach to information retrieval (abstract only). In *SIGIR*, 603.

Trotman, A. (2005). Learning to Rank. *Inf. Retr.*, *8*, 359–381.

Voorhees, E. (2002). The Philosophy of Information Retrieval Evaluation. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck (Eds.), *Evaluation of Cross-Language Information Retrieval Systems*, Volume 2406 of *Lecture Notes in Computer Science*, 143–170. Springer Berlin / Heidelberg. 10.1007/3-540-45691-034.

Xu, J. and Croft, W. B. (1998). Corpus-based stemming using cooccurrence of word variants. *ACM Trans. Inf. Syst.*, *16*(1), 61–81.

Zhai, C. and Lafferty, J. (2004). A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, *22*, 179–214.

Zhao, Y., Scholer, F., and Tsegay, Y. (2008). Effective Pre-retrieval Query Performance Prediction Using Similarity and Variability Evidence. In *ECIR'08*, 52–64.

Appendix A

Final Stemmer

The final stemmer is given in Table A.1. It is identical to that given in Table 3.8.

Table A.1: Final Stemmer

1	s	
2	d	
0	ak	
2	d	
2	ze	
0	ta	ing
3	a	oids
3	n	
2	ist	
1	an	
2	l	wsie

Appendix B

Porter Stemmer

Detailed in Porter (1980).

```
Step 1a
SSES -> SS
IES -> I
SS -> SS
S ->

   Step 1b
(m>0) EED -> EE
(*v*) ED ->
(*v*) ING ->
AT -> ATE
BL -> BLE
IZ -> IZE
(*d & not (*L or *S or *Z)) -> single letter
(m=1 & *o) -> E

   Step 1c
(*v*) Y -> I

   Step 2
(m>0) ATIONAL -> ATE
(m>0) TIONAL -> TION
(m>0) ENCI -> ENCE
(m>0) ANCI -> ANCE
(m>0) IZER -> IZE
(m>0) ABLI -> ABLE
(m>0) ALLI -> AL
(m>0) ENTLI -> ENT
```

```
(m>0) ELI -> E
(m>0) OUSLI -> OUS
(m>0) IZATION -> IZE
(m>0) ATION -> ATE
(m>0) ATOR -> ATE
(m>0) ALISM -> AL
(m>0) IVENESS -> IVE
(m>0) FULNESS -> FUL
(m>0) OUSNESS -> OUS
(m>0) ALITI -> AL
(m>0) IVITI -> IVE
(m>0) BILITI -> BLE

   Step 3
(m>0) ICATE -> IC
(m>0) ATIVE ->
(m>0) ALIZE -> AL
(m>0) ICITI -> IC
(m>0) ICAL -> IC
(m>0) FUL ->
(m>0) NESS ->
   Step 4
(m>1) AL ->
(m>1) ANCE ->
(m>1) ENCE ->
(m>1) ER ->
(m>1) IC ->
(m>1) ABLE ->
(m>1) IBLE ->
(m>1) ANT ->
(m>1) EMENT ->
(m>1) MENT ->
(m>1) ENT ->
(m>1 and (*S or *T)) ION ->
(m>1) OU ->
(m>1) ISM ->
(m>1) ATE ->
(m>1) ITI ->
(m>1) OUS ->
(m>1) IVE ->
(m>1) IZE ->
```

```
   Step 5a
(m>1) E ->
(m=1 & not *o) E ->

   Step 5b
(m > 1 & *d & *L) -> single letter
```