Evolving Lucene Search Queries for Text Classification

Laurence Hirsch
Sheffield Hallam University
Pond Street
Sheffield S1 1WB
+44 (0)114 225 5555
lauriehirsch@gmail.com

Robin Hirsch
University College London
Gower Street
London WC1E 6BT
+44 (0)20 7679 2000
r.hirsch@cs.ucl.ac.uk

Masoud Saeedi
Royal Holloway University
Egham
Surrey TW20 OEX
+44 (0)1784 434455
m.saeedi@rhul.ac.uk

ABSTRACT
We describe a method for generating accurate, compact, human understandable text classifiers. Text datasets are indexed using Apache Lucene and Genetic Programs are used to construct Lucene search queries. Genetic programs acquire fitness by producing queries that are effective binary classifiers for a particular category when evaluated against a set of training documents. We describe a set of functions and terminals and provide results from classification tasks.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]:

General Terms: Algorithms.

Keywords
text classification, Genetic Programming, Apache Lucene.

1. INTRODUCTION
Automatic text classification is the activity of assigning predefined category labels to natural language texts based on information found in a training set of labelled documents. In recent years it has been recognised as an increasingly important tool for handling the exponential growth in available online texts and we have seen the development of many techniques aimed at the extraction of features from a set of training documents, which may then be used for categorisation purposes. It has also been recognised that knowledge discovery is best served by the construction of predictive models which are both accurate and comprehensible.

In the 1980's a common approach to text classification involved humans in the construction of a classifier or 'expert system', which could be used to define a particular text category. Such a classifier would typically consist of a set of manually defined logical rules, one per category, of type

if {DNF formula} **then** {category}

A DNF ("disjunctive normal form") formula is a disjunction of conjunctive clauses; the document is classified under a category if it satisfies the formula i.e. if it satisfies at least one of the clauses. An often quoted example of this approach is the CONSTRUE system[8], built by Carnegie Group for the Reuters news agency. A sample rule of the type used in CONSTRUE to classify documents in the 'wheat' category of the Reuters dataset is illustrated below.

if ((wheat & farm) **or**
(wheat & commodity) **or**
(bushels & export) **or**
(wheat & tonnes) **or**
(wheat & winter & ¬ soft))
then
WHEAT **else** ¬ WHEAT

Such a method, sometimes referred to as 'knowledge engineering', provides accurate rules and has the additional benefit of being human understandable. That is, the definition of the category is meaningful to a human, thus producing additional uses of the rule including verification of the category. However the disadvantage is that the construction of such rules requires significant human input and the human needs some knowledge concerning the details of rule construction as well as domain knowledge [1]. Since the 1990's the machine learning approach to text categorisation has become the dominant one. In this case the system requires a set of pre-classified training documents and automatically produces a classifier from the documents. The domain expert is needed only to classify a set of existing documents. Such classifiers, usually built using the frequency of particular words in a document (sometimes called 'bag of words'), are based on two empirical observations regarding text:

1. the more times a word occurs in a document, the more relevant it is to the topic of the document.
2. the more times the word occurs throughout the documents in the collection the more poorly it discriminates between documents.

A well known approach for computing word weights is the term frequency inverse document frequency (tf-idf) weighting which assigns the weight to a word in a document in proportion to the number of occurrences of the word in the document and in inverse proportion to the number of documents in the collection for which the word occurs at least once, i.e.

$$a_{ik} = f_{ik} \log\left(\frac{N}{n_i}\right)$$

where a_{ik} is the weight of word i in document k, f_{ik} is the frequency of word i in document k, N the number of documents in the collection and n_i equal to the number of documents in which a_i occurs at least once. A classifier can be constructed by mapping a

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

document to a high dimensional feature vector, where each entry of the vector represents the presence or absence of a feature [14]; [10]. In this approach, text classification can be viewed as a special case of the more general problem of identifying a category in a space of high dimensions so as to define a given set of points in that space. This is usually accompanied by some form of feature reduction such as the removal of non-informative words (stop words) and by the replacing of words by their stems, so losing inflection information. Such sparse vectors can then be used in conjunction with many learning algorithms for computing the closeness of two documents and quite sophisticated geometric systems have been devised [3].

Although this method has produced accurate classifiers there are a number of drawbacks when compared to a rule based system.

1. The approach cannot normally identify word combinations, phrases or multi-word units e.g. 'information processing'. Only the frequency of the terms in the document is stored
2. The classifier (the vector of weights) is not human understandable.

In recent years there have been a number of attempts to produce effective classifiers that are human understandable [18]. The advantages of such a classifier include

1. The classifier may be validated by a human.
2. The classifier may be fine tuned by a human.
3. The classifier may be used for another task such as information extraction or text mining.

Generally, the attempts to produce classification systems that are human understandable have involved the production of a set of rules which are used for classification purposes. Often the set is quite large which reduces some of the qualitative advantages because it will harder for a human to comprehend the classifier. In this paper we describe a method to evolve compact human understandable classifiers using only a set of training documents. Furthermore each category in the dataset requires only one rule in the form of a Lucene search query.

The system uses genetic programming (GP) [11] to produce a synthesis of machine learning and knowledge engineering with the intention of incorporating advantageous attributes from both. The search queries produced by the GP individuals are able to use a wide variety of features including phrases, word proximity, word combinations and negative information for discrimination purposes.

2. Background

Although GP has been used in a textual environment [4][5] it has not previously been used to evolve search query classifiers for large text datasets.

2.1 Apache Lucene

In our system each GP individual in the population will produce a search query and its fitness is evaluated by applying the search query to a potentially large text dataset. With a GP population of a reasonable size evolving over 50 or more generations it is critical that such queries can be executed in a timely and efficient manner. For this reason we decided to use Apache Lucene which is an open source high-performance, full-featured text search engine. We use Lucene to build text indexes on the training and test datasets and to evaluate the queries built by the GP individuals. A full description of the indexing system is given at the official Lucene site (http://lucene.apache.org/) together with the Java source code and other useful information concerning Lucene. We currently implement a subset of the available query types as shown in **Table 1**[1].

Table 1 Lucene Query Operators

Symbol	Description
OR	The OR operator is the default operator. This means that if there is no other operator between two terms, the OR operator is used. The OR operator will retrieve a document if either of the terms exist in the text of a document.
AND	The AND operator matches documents where both terms exist anywhere in the text of a document.
NOT	Excludes documents that contain the term after NOT
+	The "+" or required operator requires that the term after the "+" symbol exist somewhere in the text of a document.
-	The "-" or prohibit operator excludes documents that contain the term after the "-" symbol.
~	Proximity searching can be used to find words are a within a specific distance of each other determined by a number following the '~'. If no number is entered then the words must occur together with no intervening text.

Lucene provides many other features and in particular a tf-idf based weighting system for search terms. However, in our application this was not used and we only collected the total number of matching documents for each search query.

3. Implementation

We summarise the key features of our implementation below

- The basic unit we use is a single stemmed word
- Lucene search queries are produced by GP individuals for each category in the dataset; thus each search query is a binary classifier
- Queries can include disjunctions, conjunctions, negations and proximity searches in Lucene query format.
- Fitness is accrued for GP individuals producing classification queries which retrieve positive examples of the category but do not retrieve negative examples. Thus the documents in the training set are the fitness cases.

3.1 Data Sets

The task involved categorising documents from the Reuters-21578 test collection which has been a standard benchmark for the text categorisation tasks throughout the last ten years. Reuters-21578 is a set of 21,578 news stories which appeared in the Reuters newswire in 1987. In our experiments we use the "ModApt´e split", a partition of the collection into a training set and a test set that has been widely adopted by text categorisation

[1] A full description of the query syntax with examples is given at http://lucene.apache.org/java/docs/queryparsersyntax.html

experimenters. We focus our discussion on the results we obtained on the top 10 (R10) categories subset although we also generated a classifier for the commonly used subset of 90 categories (R90). An in depth discussion concerning the Reuters dataset is given in [7].

3.2 Pre-Processing

Before we start the evolution of classification rules a number of pre-processing steps are made.

1. All the text is placed in lower case and stemmed.
2. A small stop set is used to remove common words with little semantic weight.
3. For each category of the dataset two Lucene indexes are created for training, one built from documents in the relevant category (positive examples) and one from the irrelevant documents (negative examples). Two further indexes from the appropriate documents in the test set are also created if testing is needed.
4. For each category of the dataset two ordered lists of features are extracted for GP use. These sets are initially created by calculating the tf-idf value for each word in the positive example category and in the negative example category. These lists are ordered according to the calculated value such that element 0 in the list has the highest tf-idf value. We will refer to these two lists as the positive features list and the negative feature list.

3.3 Fitness

GP individuals are set the task of combining selected words with special symbols to form a Lucene search query. The query is then applied to the index of positive examples and to the index of negative examples. A classification query must be evolved for each category in the training set. Each query is actually a binary classifier i.e. it will classify documents as either in the category or outside the category. The Break-Even-Point (BEP) statistics is widely used in information retrieval and text categorization. BEP finds the point where precision and recall are equal. The F1 measure is also commonly used for determining classification effectiveness and has the advantage of giving equal weight to precision and recall [17]. F1 is given by

$$F1 = \frac{2pr}{p+r}$$

where recall (r) = the number of relevant documents returned/the total number of relevant documents in the collection and precision (p) = the number of relevant documents returned /the number of documents returned. F1 also gives a natural fitness measure for an evolving classifier since BEP may favour trivial results, for example, if no data is correctly categorized then r=0 and p=1 so their average is 0.5 instead of 0 when using the harmonic average. Such classifiers are actually likely to be the norm in the early generations of a GP run, therefore, the fitness of an individual GP is assigned by calculating F1 for the query produced by that GP.

For example, if we are evolving a classifier for the Reuters 'crude' category a GP might produce the following query

barrel bbl

By default the elements of Lucene queries are disjuncts i.e. there is an implicit OR between elements of a query and the above query would retrieve any document containing either the word 'barrel' or the word 'bbl'. In fact, such a query is quite an effective classifier for the crude category and has F1 of 0.693

GP's have a tendency to 'bloat' and to produce long forms of equivalent shorter programs by including redundant sections. A typical GP produced query for the money-fx category is shown below.

dollar dollar -profit interven -profit (market AND bank) -wheat interven interven -wheat (market AND bank) -profit -profit -profit (market AND bank) dollar dollar (market AND bank) -profit –wheat -wheat

The above query is equivalent to

dollar interven (market AND bank) -wheat -profit

Also, because each element of a query is disjuncted by default there is no penalty for including elements that retrieve no documents

corn (dollar AND dollar AND NOT dollar)

will have the same fitness as the query

corn

One of the key objectives of our system was comprehensibility. We therefore applied a simple form of parsimony pressure. The resulting fitness used is the F1 of a program but where the F1 of two programs is found to be equal the shorter program is assigned a better fitness value. With this method we were able to evolve the queries shown in **Table 5** although it can be seen that they are still not necessarily in the most compact form. Comprehensibility may be improved by using other forms of parsimony pressure on the GP evolution or by using an editing program to remove redundant parts of the query.

3.4 GP Types

We use a strongly typed tree based GP [12] system with types shown in **Table 2**.

Table 2: GP Types

Type	Description
Int	An integer terminal used to identify a word
Query	A complete search query using Lucene query syntax.

In our system each GP will output a query using standard Lucene query syntax and will return a Boolean value when evaluated against a document i.e. the query will be either true or false for a particular document depending on whether the query matches the text in the document.

3.5 Terminals

We used 8 integer terminals (0 – 7).

3.6 Functions

Table 3 describes the GP functions. At the base of a GP tree we have integer terminals and at the top we have Lucene queries. A number of functions take one or more Int arguments and return a query. In this case features (normally words) are copied from either the positive or negative feature lists created in the pre-processing step. The feature returned is the feature occurring in

the list at the position defined by the Integer argument. The positive feature list is used for all function except for NOT and '–' (Lucene prohibit function see **Table 1**) which use the negative feature list. For example the query "EXISTS 1" will simply return the word at position 1 in the positive feature list. In the table we refer to a feature from the list as a 'word' which is the case at the start of the evolution.

Table 3: GP Functions

Function Name	No of args	Type of args	Return Type	Description
ADD	2	Int	Int	Add two integers.
MULT	2	Int	Int	Multiply two integers
EXISTS	1	Int	Query	Return a single word
PLUS	1	Int	Query	Return a single word with the'+' character appended to the front.
MINUS	1	Int	Query	Return a single word with the'-' character appended to the front.
OR	2	Query	Query	Return a query by concatenating the two query arguments and inserting a space between them.
AND	2	Int	Query	Return in a query in the form "(word0 AND word1)"
NOT	2	Query	Query	Return a query in the form (query1 NOT query2)
DNF	3	Int	Query	Return a query in the form "(word0 AND word1 AND NOT word2)"
PHRASE	2	Int	Query	Return a query in the form "word0 word1"~ (Lucene proximity search: see **Table 1**)
NEARX	3	Int	Query	Return a query in the format of PHRASE but with the value of the third integer argument appended at the end

3.7 Example Programs

To illustrate the system in action we show three Genetic Programs below which were evolved using the Reuters 'crude' category as positive examples and the Lucene search queries which they output.

Query: oil
Tree: (EXISTS 0)

Query: (price AND oil)
Tree: (AND (ADD 1 2) 0)

Query: (opec AND petroleum AND NOT year) (oil AND barrel) "crude price"~5
Tree: (OR (DNF (ADD 4 0) 5 3) (OR (AND 0 1) (NEARX 2 3 5)))

3.8 GP Parameters

We used the ECJ system (http://cs.gmu.edu/~eclab/projects/ecj) and a fixed set of GP parameters summarised in **Table 4**

Table 4 GP Parameters

Parameter	Value
Population	1024
Generations	50
Typing	Strongly typed
Creation Method	Ramped half and half
GP format	Tree Based
Selection type	Tournament
Tournament size	7
Termination	Perfect classifier or max gen
Mutation probability	0.1
Reproduction probability	0.1
Crossover probability	0.8
Elitism	No
ADF	No
Max tree depth at creation	9
Max tree depth for crossover	17
Max tree depth for mutation	17
Subpopulations	3
Number of runs	5

3.9 Making use of discovered queries

The system we describe has the advantage of making extensive use of negative data. We have extended this capability by storing information from successfully evolved queries for use in classifying other categories. This is done in the following way.

1. At the end of evolution for a particular category the best query is selected.
2. Positive words and phrases are extracted from the query and stored in the negative features list.
3. All the new elements are assigned a score by multiplying the number of positive documents in the category by the F1 (training) measure of the query from which they were extracted, so that the most useful will appear at the top of the negative feature list.

GP individuals can then make use of this data when evolving queries for the remaining categories. The data is kept on the list so that it may also be used during later GP runs.

4. Experiments

4.1 Objectives

The objectives of our experiments were two fold:

1. To evolve effective classifiers against the text datasets.

2. To automatically produce compact and human understandable classifiers in search query format.

4.2 Evolution

Figure 1 shows a fairly typical pattern of evolution, where precision (*p*) and recall (*r*) may move up and down but there is a general improvement in the F1 measure (training data)..

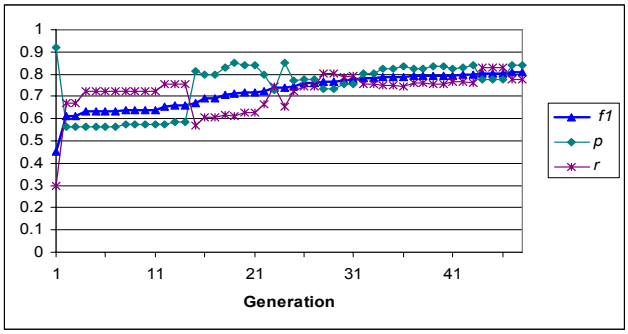

Figure 1 Best query by generation (R10 category Ship).

In all the experiments reported here the GP system only had access to the training data. The final result was obtained using the queries produced during the training against the test data.

4.3 Example of Query Generation

In this section we included an example of the evolution of a rule for the R10 category "corn". Some good results can be evolved quite easily on this category e.g. an individual producing the one word query

corn

appears in generation 0 and will achieve a fitness (F1 against the training data) of 0.756. In generation 7 a new query is found giving a fitness of 0.77

(-compan AND corn)

This query searches for documents which do not contain a word starting with the string 'compan' and do contain the string 'corn'. In generation 11 the following query is found

maiz corn

This obtains a fitness of 0.862 and will retrieve documents which include the words 'maiz' or 'corn'. In generation 19 this is replaced by the slightly more successful query

maiz -bank corn

which obtains a fitness of 0.896 and will retrieve documents which do not contain the word 'bank' but do contain either the word 'maiz' or the word 'corn'.

4.4 Performance

GP systems are notoriously computationally intensive and unfortunately the system described here is no exception. All the experiments were run on a Pentium p3 processor running at 1GHz and with 512M of memory. To generate the R10 queries shown in **Table 5** took over 5 hours.

Every GP must query two Lucene indexes to obtain its fitness value and training must occur on every category of the dataset. For each category we have implemented 5 GP runs of 50 generations each. Therefore for the R10 case the training involved the execution of a possible 2 * 1024 * 50 * 5 * 10 = 5120000 Lucene queries. This is on top of the normal operations required for the GP system such as crossover, mutation and population management. In light of this the training time is actually a testament to the efficiency of Lucene and ECJ.

The good news is that most of this effort goes into increasing the classification accuracy by only a small amount. For example, a reasonable classifier for the R10 with a macro average F1 of 0.749 and micro average of 0.816 can be generated in 4 minutes. In this case a population of 128 GPs evolved using only 1 run and 20 generations. Furthermore there is ample opportunity for parallelization of many parts of the system, for example, the GP runs, the evaluation of separate categories and subpopulation evolution are all open to simple forms of parallelization.

The result of all the training work is a search query. To test the R10 classifier requires the execution of 10 search queries and the result will occur in a time frame well below human perception. The fact that search queries will scale up to large text databases, such as the Internet, is well known.

4.5 Overfitting

As expected we found overfitting to be most severe when the training data was quite limited. For example on the R90 set the classifier

kerosen (jet AND logist AND NOT qtr) "paralyz javier"~6

evolved and was a perfect classifier for the 5 training examples but failed to match any of the test documents.

Figure 2 shows the closeness in F1 test and training values for classifiers of the R90 dataset in relation to training data size.

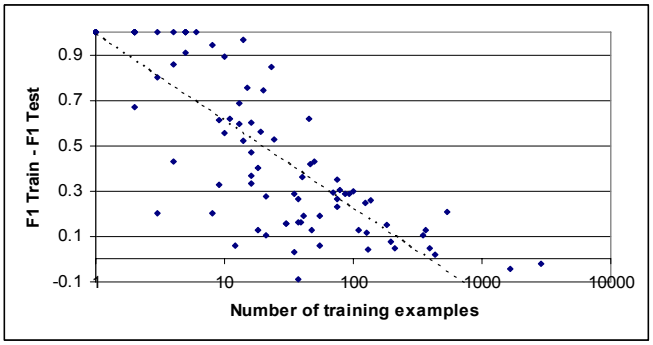

Figure 2 Difference between F1 train and F1 test (R90)

Perfect classifiers of the training data are commonly evolved where the set of positive training documents is small e.g. less that 20 documents. However we never evolved such a classifier for any of categories of the R10.

5. Results

A classification rule was evolved for each category by using 5 GP runs for each category and selecting the best query, as measured against the training data, to emerge from the 5 runs. The selected query was then run against the test set to produce the final result. The query produced is an important part of our system since we are emphasising the qualitative difference of this particular classifier, and so we give the complete set of classification search queries for the R10 categories dataset.

Table 5 R10 Classification Queries

Class	F1 (test)	Search Query
Acq	0.913	-vs -ct -tonn inc corp "dlr share"~ -"wheat wheat"~20 inc ((bui AND compani AND NOT shr) NOT "inc share"~10) -"barrel barrel"~20 inc -dollar bid -rate acquir -ct merger corp sell -"march loss"~10 corp inc corp -"japanes trade"~20 -ship corp -"rate bank"~10 -ct takeov -rate inc -grain stake -"march loss"~10 corp -"trade export"~20 acquir -"barrel barrel"~20 -qtr acquir acquisit
Corn	0.896	maiz –bank corn
Crude	0.87	(((barrel AND barrel AND NOT acquisit) (-stake -stake -trade (-stake -wheat -"compani share"~10 (+oil NOT compani) NOT acquisit) –crop crude "oil petroleum"~20) NOT "march loss"~10) NOT net) (+barrel NOT merger) NOT dollar) -"march loss"~10 ("opec oil"~5 (ga AND import) -"rate bank"~10 (refineri AND oil) -"corn corn"~10 "oil energi"~20 (barrel AND barrel AND NOT acquisit) "oil petroleum"~20 "oil energi"~20 "oil explor"~5 (ga AND opec) -vs
Earn	0.958	("vs vs"~20 NOT takeov) "ct vs"~20 "march qtr"~5 (rev AND shr) "qtr qtr"~20 "ct vs"~20 ("vs vs"~20 NOT acquir) ("ct vs"~20 "march qtr"~5 (loss AND net AND NOT takeov) -acquir (dividend AND dividend AND NOT stake) (("net loss"~0 NOT acquir) NOT net) "ct year"~20 ("vs vs"~20 NOT acquir) "qtr qtr"~20 (ct AND qtr) ((("vs vs"~20 NOT merger) (profit AND year AND NOT stake) NOT acquir) NOT acquir) (profit AND year AND NOT corp) "year net"~20 payabl NOT net) "ct year"~20 "ct vs"~20 "march qtr"~5 (profit AND net AND NOT takeov) (dividend AND year AND NOT stake) (ct AND qtr) ("ct vs"~20 "march qtr"~5 (rev AND shr) "ct shr"~20 ("vs vs"~20 NOT acquir) "qtr qtr"~20 NOT acquir) (profit AND five AND NOT corp) "year net"~20 "ct vs"~20 (net AND shr AND NOT stake) (profit AND net AND NOT corp) "qtr qtr"~20 "ct vs"~20 (ct AND qtr) ("net loss"~0 NOT acquir) "year net"~20 -"wheat wheat"~20 "ct vs"~20 -bid (profit AND net AND NOT takeov) "mln ct"~21 (dividend AND dividend AND NOT stake) -"trade export"~20 (profit AND ct AND NOT corp) "ct vs"~20 "march qtr"~5
Grain	0.933	wheat "rice tonn"~20 "agricultur tonn"~20 maiz "cereal week"~20 (grain AND agricultur) wheat barlei (agricultur AND corn) "cereal state"~20 wheat wheat (grain (corn AND corn AND NOT "ct vs"~20) (grain AND agricultur) wheat -inc barlei crop "rice tonn"~20 wheat -inc barlei crop -bank (maiz AND agricultur) wheat grain maiz "cereal week"~20 wheat barlei wheat (corn AND corn AND NOT "ct vs"~20) crop "export rice"~20 NOT compani)
Interest	0.723	"pct outright"~20 -shr "rate prime"~20 "rate discount"~20 -shr "rate prime"~20 "rate bank"~20 "pct outright"~20 "rate bank"~20 "feder monei"~20 (fed AND pct) "rate discount"~20 "monei central"~30 "pct outright"~20 "rate bank"~20 (discount AND england AND NOT "ct year"~20) -shr "rate prime"~20 "prime point"~20 "rate prime"~20 -stake "central monei"~20 "england pct"~20 "pct outright"~20 "rate discount"~20 "rate prime"~20 "rate bank"~20 -dividend "rate prime"~20 "discount bank"~5 -commission -shr "prime point"~20 "outright england"~4 "monei rate"~ "down england"~4
Money-fx	0.734	"currenc dollar"~20 -wheat "currenc currenc"~18 "currenc currenc"~5 -crop currenc "financ dollar"~7 -"year net"~20 "monei market"~0 "currenc currenc"~5 "pari dollar"~20 "financ dollar"~20 -dividend (-net NOT "ct year"~20) "currenc currenc"~5 "sai dollar"~20 "london dollar"~20 "currenc far"~5 -qtr (market AND fed AND NOT "year net"~20) "currenc currenc"~5 -"year net"~20 "bank dollar"~20 "bank dollar"~20 "rate market"~0 -commiss "dollar market"~ "chang dollar"~7 "england currenc"~20 "dealer dollar"~20 "monetari dollar"~20 "monetari rate"~20 -"oil crude"~20
Ship	0.840	(+vessel NOT "march qtr"~5) "sea port"~20 (warship (sao (("freight ship"~5 NOT "march qtr"~5) "sai port"~20 NOT "march qtr"~5) "hour port"~20 NOT "march qtr"~5) "passeng port"~20 (-"currenc far"~5 -corn ((+vessel NOT "march qtr"~5) "cargo port"~20 NOT "march qtr"~5) -net vessel -per "vessel gulf"~ (tonn AND freight) -shr (freight AND sea) "transport tanker"~20 -"currenc monei"~20 tanker (((+vessel NOT "march qtr"~5) "tanker port"~20 NOT "march qtr"~5) NOT "march qtr"~5) "vessel gulf"~ (-tariff -wheat ship "cargo water"~3 "vessel gulf"~ "port strike"~20 -"profit year"~20 ship NOT "march qtr"~5) NOT "march qtr"~5) NOT shr)
Trade pr	0.795	"deficit trade"~73 "deficit surplu"~7 ("deficit trade"~73 ((sanction AND japanes) NOT payabl) -monei (trade AND retali AND NOT "rice tonn"~20) (sanction AND japanes) -payabl "deficit trade"~30 ((sanction AND japanes) NOT payabl) ((sanction AND japanes) NOT payabl) ("surplu japanes"~20 -grain "deficit trade"~7 NOT "currenc dollar"~20) "surplus trade"~7 -corn NOT "barrel barrel"~20) ((sanction AND japanes) NOT payabl) ((sanction AND japanes) NOT payabl) (("surplu japanes"~20 -grain "trade polici"~20 (trade AND sanction AND NOT "ct shr"~20) NOT "currenc dollar"~20) -monei "deficit trade"~3 "deficit trade"~73 -payabl "trade retali"~20 "trade minist"~20 ((sanction AND japanes) NOT payabl) (surplu AND deficit AND NOT dividend) ("trade tariff"~20 -grain (import AND trade AND NOT payabl) -corn NOT vessel) "surplus trade"~7 -corn NOT "barrel barrel"~20) -"currenc dollar"~20
Wheat	0.886	((((((((((wheat NOT "year net"~20) NOT earn) NOT secur) NOT acquir) NOT "profit year"~20) NOT "monetari rate"~20) NOT "qtr qtr"~20) NOT "qtr qtr"~20) NOT treasuri) NOT payabl)

Table 6 shows classifiers R10 dataset. In this case we show the BEP as in the past this result has been the most widely used and is therefore useful for comparison purposes. We are particularly interested in the other rule based classifiers which are at least partly human understandable. These are TRIPPER [18], RIPPER [6], ARC-BC (3072 rules) [2] and C4.5 [13]. The results for bigrams are from [16] and the results of other classifiers are taken from [10]. We also note that the query evolved for the wheat category which scores and F1 of 0.886 outperforms the human constructed rule discussed in the introduction which has an F1 of 0.84

Table 6 Comparison by category (R10)

BEP	*GP-TC*	*trip*	*rip*	*ARC-BC*	*C4.5*	*bi-gram*	*Bayes*	*Roc-chio*	*k-NN*	*SVM (poly)*	*SVM (rbf)*
acq	91.3	86.3	85.3	89.9	85.3	73.2	91.5	92.1	92.0	94.5	**95.2**
corn	**90.6**	85.7	83.9	82.3	87.7	60.1	47.3	62.2	77.9	85.4	85.2
crude	87.3	82.5	79.3	77.0	75.5	79.6	81.0	81.5	85.7	87.7	**88.7**
earn	95.9	95.1	94.0	89.2	96.1	83.7	95.9	96.1	97.3	98.3	**98.4**
grain	**93.3**	87.9	90.6	72.1	89.1	78.2	72.5	79.5	82.2	91.6	91.8
interest	72.3	71.5	58.7	70.1	49.1	69.6	58.0	72.5	74.0	70.0	**75.4**
money	73.4	70.4	65.3	72.4	69.4	64.2	62.9	67.6	**78.2**	73.1	75.4
ship	84.3	80.9	73.0	73.2	80.9	69.2	78.7	83.1	79.2	85.1	**86.6**
trade	**79.5**	58.9	68.3	69.7	59.2	51.9	50.0	77.4	77.4	75.1	77.3
wheat	**90.1**	84.5	83.0	86.5	85.5	69.9	60.6	79.4	76.6	84.5	85.7
macro-avg	85.8	80.4	78.1	78.2	77.8	70.0	69.8	79.1	82.1	84.5	**86.0**

The results for the R10 set show that GPTC produces rules of higher accuracy than any other rule based system in every category. We should note that GPTC produces only one rule per category as opposed to hundreds or thousands using some of the other rule based methods [2]. The comprehensibility of the GPTC queries is quite variable and some are perhaps too complex for a non-expert to deal with. However we suggest that the readability of GPTC queries does compare favourably to other rule based systems which often include large sets of rules.

Unfortunately we do not have the micro average available for all the systems shown inTable 6, however Table 7 shows the results for GPTC against a recent survey of over 40 classifiers used for the Reuters set [7]. The results show that GPTC to be well above average in the task of classifying the R10 set but somewhat below average when classifying the R90 set.

Table 7 Reuters Comparison 2

	Microaveraged F1		**Macroaveraged F1**	
	GPTC	**Survey Average**	**GPTC**	**Survey Average**
R(10)	0.897	0.852	0.847	0.715
R(90)	0.772	0.787	0.418	0.468

6. Future Work

We are investigating the usefulness of new GP functions using numeric terminals for identifying frequency information. Functions such as '>' return a Boolean value based on the frequency of a particular word in a document [2].

We would like to run the classifier on a larger dataset such as the full Ohsumed set or the Reuters RCV1-V2 set. This would require an upgrade in hardware resources and ideally a parallel implementation as discussed above.

We believe that the system described here may be of particularly value when used in conjunction with other classification systems in a classification committee [15].

7. Conclusion

We have produced a system capable of generating classification search queries with no human input beyond the identification of training documents which are useful to the task of discriminating between text documents. The classifier makes use of conjunction, disjunction, negation and word proximity. We believe this new arrangement for a text classifier has important advantages stemming from its compactness, its comprehensibility to humans and its search query format.

We suggest that there may be a number of areas within automatic text analysis where the technology described here may be of use.

8. References

[1] Apt´e, C., F. J. Damerau, and Weiss, S. M. 1994. Automated learning of decision rules for text categorization. *ACM Trans. on Inform. Syst. 12*, 3, 233–251.ATTARDI

[2] Antonie, M. L. and Zaane, O. R. *Text document categorization by term association*. In IEEE International Conference on Data Mining, pages 19--26, December 2002

[3] Bennet, K., Shawe-Taylor, J. and Wu, D. 2000. Enlarging the margins in perceptron decision trees. *Machine Learning* 41, pp 295-313

[4] Bergström, A., P., Jaksetic and Nordin, P. 2000. Enhancing Information Retrieval by Automatic Acquisition of Textual Relations Using Genetic Programming. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces* (IUI-00), pp. 29-32, ACM Press.

[5] Clack, C., Farrington, J., Lidwell, P. and Yu, T. 1997. Autonomous Document Classification for Business, in *Proceedings of the ACM Agents Conference.*

[6] Cohen, W. 1995 Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123

[7] Debole, F. and Sebastiani, F 2005. *An analysis of the relative hardness of Reuters-21578 subsets***.** Journal of the American Society for Information Science and Technology, 56(6):584-596.

[8] Hayes, P. J., Andersen, P. Nirenburg, I. and Schmandt, L. M. 1990. Tcs: a shell for content-based text categorization. In *Proceedings of CAIA-90, 6th IEEE Conference on Artificial Intelligence Applications* (Santa Barbara,CA, 1990), 320–326.

[9] Hirsch L., Saeedi M. and Hirsch R., *Evolving Text Classification Rules with Genetic Programming* Applied Artificial Intelligence, (AAI 19/7*),* Taylor & Francis, August 2005

[10] Joachims, T. 1998. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of the l0th European Conference on Machine Learning* (ECML98), pp 137-142.

[11] Koza, J.R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge MA

[12] Montana, D. 1995. Strongly Typed Genetic Programming. *In Evolutionary Computation*. 3:2, 199--230. The MIT Press, Cambridge MA.

[13] Quinlan, J. R. *Bagging, boosting, and C4.5*. In *Proceedings, Fourteenth National Conference on Artificial Intelligenc*e, 1996.

[14] Salton,G., Singhal, S., Buckley, C. and Mitra, M. 1996. Automatic Text Decomposition Using Text Segments and Text Themes. In *Proceedings of the hypertext '96 Conference*, Washington D.C. USA.

[15] Sebastiani, F. 2002. Machine learning in automated text categorization, *ACM Computing Surveys*, 34(1), pp. 1-47.

[16] Tan, C.M., Wang, Y.F., and Lee, C.D. 2002. The use of bigrams to enhance text categorization In *Information Processing and Management: an International Journal*, Vol 38, Number 4 Pages 529-546

[17] Van Rijsbergen, C.J. 1979. *Information Retrieval*, 2nd edition, Department of Computer Science, University of Glasgow

[18] Vasile, F., Silvescu, A., Kang, D-K. and Honavar, V. 2006. TRIPPER: An Attribute Value Taxonomy Guided Rule Learner. *Proceedings of the Tenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Berlin: Springer-Verlag. pp. 55-59