Adaptive Information Filtering: improvement of the matching technique and derivation of the evolutionary algorithm

D.R. Tauritz and I.G. Sprinkhuizen-Kuyper
LIACS, Leiden University
{dtauritz,kuyper}@cs.leidenuniv.nl
http://www.wi.leidenuniv.nl/home/{dtauritz,kuyper}

April 1999

Abstract

Adaptive Information Filtering is concerned with filtering information streams in changing environments. The changes may occur both on the transmission side (the nature of the streams can change) and on the reception side (the interests of a user can change). The research described in this report details the progress made in a prototype Adaptive Information Filtering system based on weighted trigram analysis and evolutionary computation. The main improvements of the algorithms employed by the system concern the computation of the distance between weighted trigram vectors and a further analysis of the two-pool evolutionary algorithm. We tested our new prototype system on the Reuters-21578 text categorization test collection.

Contents

1 Introduction

We live in what is often termed the “information age”. It might more appropriately be called the “data age”, for only relevant data is information, and finding relevant data among the ever greater accumulations of available data is becoming increasingly more difficult. One of the fields dealing with this problem is Information Filtering (IF). IF is the process of filtering data streams in such a way that only particular data are preserved, depending on certain information needs. The IF environment is the combination of data stream and information needs. When the data stream and the information needs are not changing over time the IF environment is said to be static. When, however, the IF environment is dynamic, as opposed to static, an adaptive information filtering (AIF) system is called for. An AIF system is an IF system capable of adapting to changes in both the data stream and the information needs.

One of the essential ingredients in any information retrieval (IR) or IF system is its ability to match a query (in the case of an IR system) or a profile (in the case of an IF system) with the documents available for perusal. While optimally a semantical match should be performed, that is not currently feasible and we have to be satisfied with a syntactical match. A good general reference to the field of IR/IF is [4].

The most widely employed syntactical representation of textual documents is based on term indexing (see for example [6]). In manual indexing keywords are manually assigned to a document, while in automatic indexing the frequencies of all the terms occuring in a document are indexed. Term indexing has several drawbacks, such as its sensitivity to spelling variations and errors, its static nature (the terms need to be known beforehand which is fine for IR but not for IF) and its reliance on linguistic preprocessing, such as stop word removal and word stemming, to make it effective.

Another approach which in the last decade has received quite a bit of attention is based on the so-called n-gram analysis [3]. The n stands for a positive integer. Application of n-gram analysis produces an n-gram frequency vector which holds the frequencies of all the distinct character combinations of length n. In 1-gram analysis the occurrence of single letters is determined, in 2-gram analysis that of pairs of letters, in 3-gram analysis that of triplets, etc. When talking about a specific value of n, especially for lower values of n, often its Latin name is used instead of the numeric value, so 2-grams are often called bi-grams or bigrams, 3-grams trigrams, 4-grams quadgrams, but 7-grams usually just 7-grams. For example, the word “coconut” consists of the bigrams “co”, “oc”, “on”, “nu” and “ut”, all with a frequency of one except for “co” which has a frequency of two. The trigrams are “coc”,“oco”,“con”,“onu” and “nut”, all with a frequency of one. The use of n-gram analysis has many advantages over term-based systems, such as being more robust when dealing with spelling variations or errors and not requiring linguistic preprocessing which facilitates the deployment of n-gram-based systems in multi-topic or multi-language environments [2]. However, also an n-gram-based system can potentially benefit from preprocessing, since for example when the stop word ‘the’ is removed, the trigram ‘the’ becomes of significance.

In [9] it was shown that term indexing — traditionally used in IR/IF systems — is in general not suited for AIF, but that weighted trigram analysis is. See [8] for an example of a term-based AIF system for use in a restricted domain. A prototype AIF system based on weighted trigram analysis was introduced in [9] and [10]. For $n < 3$ n-gram analysis does not provide sufficient syntactical information [7] and for $n > 3$ advanced sparse vector representations are required which will be employed in future versions of our AIF system.

The matching technique used in the original prototype AIF system was based on the Euclidean metric, which is a special case of the Minkowski ℓ_p-metric, namely for p equal to two (p equal to one is called the Manhattan metric). This report details the advances made in the matching technique. An important improvement is normalizing the weighted trigram vectors instead of the trigram vectors themselves. It also introduces the Manhattan metric as a possible alternative to the Euclidean metric in the prototype AIF system. For a general introduction to measurements in information science see [1].

A crucial step in working with weighted trigram analysis is to find the right weight vector. Our first prototype AIF system introduced a novel two-pool evolutionary algorithm (EA) for optimizing weight vectors. EAs are a class of optimization algorithms which come in handy when no a-priori solutions to a specific optimization problem are available. They work by evolving a population of trial solutions using techniques inspired by evolutionary biology. For an easy introduction to evolutionary computation (EC) see chapter 4 of [9]; for a more comprehensive introduction to EC see [5]. This report provides a full derivation of the two-pool EA, showing that it is a special case of a whole family of classification EAs.

A new prototype AIF system based on the improved matching technique has been constructed. This report describes the new system and presents the results of testing it on the Reuters-21578 text categorization test collection. Using a standard test collection will facilitate comparing these results with other case studies. The Reuters collection has embedded tags indicating common usage in text categorization tests. They were not suitable for our purposes which prevents our results from being compared to previous studies which did employ those tags. However, as the collection is readily available and later in this report we describe how we obtained the training and test sets for our research, the code for which follows in appendix A, we facilitate conducting studies which can be compared to our results.

The report is structured as follows. In section 2 we give a global description of the complete system. In section 3 we describe the distance measures. The details of the two-pool EA are presented in section 4. In section 5 our new prototype AIF system is explained, while section 6 describes the Reuters-21578 test collection and the results of our experiments with that collection. Finally, section 7 gives our conclusions, while two appendices contain C++ code for tools we used in our experiments.

2 Overview of the AIF system

This section is meant to illustrate the working of the system as a whole without drowning the reader in all the details which are given later in this report. The core of the system is the clustering cycle (see figure 1). The clustering algorithm uses a weight vector to compare the trigram frequency vector of a document with the prototype vectors of the clusters and decides in what cluster the document will be classified. Depending on the parameters of the cluster algorithm, the prototype vector of the chosen cluster will shift a bit in the direction of the newly presented document vector. The prototype vectors are initialized by averaging the trigram vectors of a number of documents belonging to each cluster (class).

The weight vector and the parameters of the cluster algorithm (the cluster radius and the shift factor) are determined by the EA. So the EA works on a population of individuals each containing a chromosome with genes existing of the components of the weight vector and the parameters of the cluster algorithm. The fitness of an individual is determined by dividing the number of documents it has correctly classified by the total number of documents it has classified.

3 Measuring distance in weighted trigram frequency vector space

The performance of a matching technique is called its discriminating power. The higher the discriminating power, the better a technique is able to separate documents which are semantically dissimilar and to group together

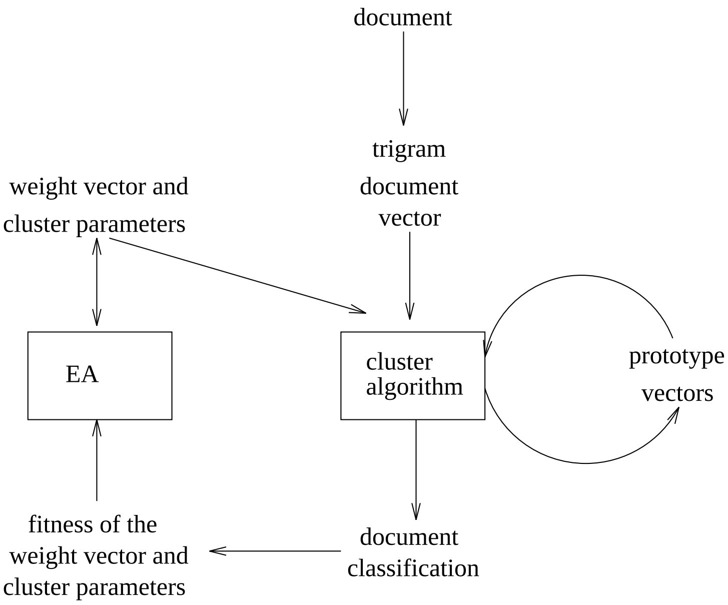

Figure 1: Schematic overview of the adaptive IF system.

documents which are semantically similar. In [9] it was shown that the combination of weighted trigram analysis (each trigram is assigned a weight indicating its relative importance) and the Euclidean distance metric has sufficient discriminating power for document classification.

The size of the alphabet used will be indicated with $|a|$. Consider two document vectors d_1 and d_2. Let $f = (f_1, f_2, \cdots, f_n)$ and $g = (g_1, g_2, \cdots, g_n)$ with $n = |a|^3$ be the corresponding trigram frequency vectors for these documents. Let $w = (w_1, w_2, \cdots, w_n)$ with $w_i \geq 0$, $i = 1, \cdots, n$ be the weight vector giving the relative importance of the different trigram frequencies. The weighted trigram vectors $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$ corresponding to f and g respectively are defined as follows: $x_i = f_i w_i$ and $y_i = g_i w_i$ for $i = 1, \cdots, n$.

In [9] it was argued that the trigram frequency vectors had to be normalized to prevent the length of a document influencing the distance metric. This was accomplished by introducing $\tilde{f} = (\tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_n)$ with $\tilde{f}_i = f_i / \sum_{j=1}^{n} f_j$ and $\tilde{g} = (\tilde{g}_1, \tilde{g}_2, \cdots, \tilde{g}_n)$ with $\tilde{g}_i = g_i / \sum_{j=1}^{n} g_j$ and defining $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_n)$ and $\tilde{y} = (\tilde{y}_1, \tilde{y}_2, \cdots, \tilde{y}_n)$ as follows: $\tilde{x}_i = \tilde{f}_i w_i$ and $\tilde{y}_i = \tilde{g}_i w_i$. The match between d_1 and d_2 was estimated by applying the Euclidean distance metric to $\tilde{x}$ and $\tilde{y}$:

$$\rho(\tilde{x}, \tilde{y}) = \sqrt{\sum_{i=1}^{n} (\tilde{x}_i - \tilde{y}_i)^2} \tag{1}$$

However, in [9] the weights were not normalized, which allowed the following to happen (for simplification an alphabet of four symbols will be used for this and all subsequent examples):

> Proportional weight vector example
>
> In this example the weight vectors are proportional. And, since the weights indicate relative importance, we want the result to be the same for both weight vectors. Given the normalized trigram frequency vectors $\tilde{f} = (0.2, 0.0, 0.3, 0.5)$ and $\tilde{g} = (0.4, 0.0, 0.6, 0.0)$ and weight vector $w = (1, 2, 1, 2)$, the weighted trigram vectors are $\tilde{x} = (0.2, 0.0, 0.3, 1.0)$ and $\tilde{y} = (0.4, 0.0, 0.6, 0.0)$. This yields the Euclidean distance $\rho(\tilde{x}, \tilde{y}) \approx 1.063$. With the weight vector $w = (2, 4, 2, 4)$ the weighted trigram vectors are $\tilde{x} = (0.4, 0.0, 0.6, 2.0)$ and $\tilde{y} = (0.8, 0.0, 1.2, 0.0)$. This yields a Euclidean distance of $\rho(\tilde{x}, \tilde{y}) \approx 2.126$. The results are not the same because the weighted vectors were not normalized.

We want the weighted distribution to influence only the distance, not the actual sizes of the weights — just as we want the trigram frequency distribution to influence the distance, not the actual sizes of the frequencies. It would appear that normalizing the weights will solve this problem. This can be accomplished by introducing $\tilde{w} = (\tilde{w}_1, \tilde{w}_2, \cdots, \tilde{w}_n)$ with $\tilde{w}_i = w_i / \sum_{j=1}^{n} w_j$ and defining $\hat{x} = (\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$ and $\hat{y} = (\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n)$ as follows: $\hat{x}_i = \tilde{f}_i \tilde{w}_i$ and $\hat{y}_i = \tilde{g}_i \tilde{w}_i$. The match between d_1 and d_2 can then be estimated by applying the Euclidean distance metric to $\hat{x}$ and $\hat{y}$:

$$\rho(\hat{x}, \hat{y}) = \sqrt{\sum_{i=1}^{n} (\hat{x}_i - \hat{y}_i)^2} \tag{2}$$

But there are still more problems lurking in the woods. Consider for instance the results of a slightly modified version of the above example using normalized trigram frequency vectors and a normalized weight vector:

> Normalized weight vector example (1)
>
> If the fourth normalized weight is zero, only the first three trigrams are considered, and as both trigram frequency vectors convey the same information about the trigrams being considered, namely that the third trigram is twice as prevalent as the first one, the distance between them should be zero. Given the normalized trigram frequency vectors $\tilde{f} = (0.2, 0.0, 0.3, 0.5)$ and $\tilde{g} = (0.4, 0.0, 0.6, 0.0)$ then using the normalized weight vector $\tilde{w} = (0.3, 0.4, 0.3, 0.0)$, the weighted trigram vectors are $\hat{x} =$

> $(0.06, 0.0, 0.09, 0.0)$ and $\hat{y} = (0.12, 0.0, 0.18, 0.0)$. This yields the Euclidean distance $\rho(\hat{x}, \hat{y}) \approx 0.108$. The distance is not zero, indicating that there is still a flaw in the matching technique.

Another problem is illustrated by the following example:

> Normalized weight vector example (2)
> If the second normalized trigram frequency is zero for both trigram vectors, only the first, third and fourth normalized trigram frequencies are considered. And if both normalized weight vectors convey the same information about the trigrams being considered, namely that the fourth is twice as important as the first and the third, the distance should be the same for both normalized weight vectors. Given the normalized trigram frequency vectors $\tilde{f} = (0.2, 0.0, 0.3, 0.5)$ and $\tilde{g} = (0.4, 0.0, 0.6, 0.0)$ and normalized weight vector $\tilde{w} = (0.1, 0.6, 0.1, 0.2)$, the weighted trigram vectors are $\hat{x} = (0.02, 0.0, 0.03, 0.1)$ and $\hat{y} = (0.04, 0.0, 0.06, 0.0)$. This yields the Euclidean distance $\rho(\hat{x}, \hat{y}) \approx 0.106$. With the normalized weight vector $\tilde{w} = (0.2, 0.2, 0.2, 0.4)$ the weighted trigram vectors are $\hat{x} = (0.04, 0.0, 0.06, 0.2)$ and $\hat{y} = (0.08, 0.0, 0.12, 0.0)$. This yields the Euclidean distance $\rho(\hat{x}, \hat{y}) \approx 0.213$. The distances are not the same, again indicating a flaw in the matching technique.

In the last two examples the indicated flaw is caused by one and the same mistaken assumption, that is, that we can normalize the trigram frequency vectors independently from the weight vectors. If we want to measure the distance between two weighted trigram frequency vectors then those are the vectors that need to be normalized. This can be accomplished by introducing $\overline{x} = (\overline{x}_1, \cdots, \overline{x}_n)$ with $\overline{x}_i = x_i / \sum_{j=1}^{n} x_j$ and $\overline{y}_i = y_i / \sum_{j=1}^{n} y_j$.

The match between d_1 and d_2 can then be estimated by applying the Euclidean distance metric to $\overline{x}$ and $\overline{y}$:

$$\rho(\overline{x}, \overline{y}) = \sqrt{\sum_{i=1}^{n} (\overline{x}_i - \overline{y}_i)^2} \tag{3}$$

Applying this to the last two examples produces the correct results.

An alternative to the Euclidean metric is the Manhattan metric. Using it the match between d_1 and d_2 can be estimated as follows:

$$\rho'(\overline{x}, \overline{y}) = \sum_{i=1}^{n} |\overline{x}_i - \overline{y}_i| \tag{4}$$

4 Applying evolutionary computation to classification

In our AIF system the classification of a document vector is dependent on the weight vector being used. We determine this vector by using an evolutionary algorithm (EA). In this section we will consider the development of classification EAs (CEAs) more generally, but for our concrete system the members of a population are weight vectors, the score of a member is the number of correctly classified documents and its age is the total number of documents it has classified.

The set of objects to classify will be denoted with S and the number of objects in S with $|S|$. For short σ will stand for an object and $c(\sigma)$ for the class σ maps to. The set $P = \{P_1, P_2, \cdots, P_{pop\_size}\}$ is the population of trial solutions with *pop_size* a positive integer. For the purpose of indexing the population members we define i as an integer between 1 and *pop_size*. Two essential components of any CEA are the evaluation of all the population members and, based on that, the evolvement of the population. The evolvement component will be denoted with $EVOLVE(P)$. The evaluation component will be denoted with $EVAL(S,P)$ and is defined as follows:

$$EVAL(S,P): \quad \forall P_i \in P \text{ determine } FITNESS(S,P_i) \tag{5}$$

The fitness of a trial solution given an object set is the average score of that trial solution on classifying all the objects in the object set. The range of the fitness is from zero to one with zero being the worst (all classifications incorrect) and one the best (all classifications correct). The fitness function is defined as follows:

$$FITNESS(S,P_i) = \frac{\sum_{\forall \sigma \in S} RESULT(\sigma, P_i)}{|S|} \tag{6}$$

The result of classifying an object given a trial solution is either zero (incorrect) or one (correct). The result function is defined as follows:

$$RESULT(\sigma, P_i) = \begin{cases} 0 & \text{if } classify(\sigma, P_i) \neq c(\sigma) \\ 1 & \text{if } classify(\sigma, P_i) = c(\sigma) \end{cases} \tag{7}$$

The result function works by comparing the actual mapping of an object to the mapping of that object computed using a trial solution. The function which performs that computation is defined as:

$$CLASSIFY(\sigma, P_i) = \text{the class } \sigma \text{ maps to using } P_i \tag{8}$$

The CEA can then be defined as given in Algorithm 1.

Algorithm 1 Static object set

```
initialize $S$, $P$
$EVAL(S,P)$
while (not termination condition) do
    $EVOLVE(P)$
    $EVAL(S,P)$
end
```

4.1 Expanding object set

If S expands in time we can simply execute Algorithm 1 after each expansion to find a mapping from object space to class space at any given time. If the set of objects is smaller than the object space and represents it better as it expands, then the mapping found by the CEA will better approximate the mapping from object space to class space as time progresses. In this case it is likely that the mapping found at any particular time is a good approximation of the mapping to be found the following time and therefore would make a good starting point for the next search. Time will be denoted with τ and the object added to S at $\tau = \hat{\tau}$ with $\sigma_{\hat{\tau}}$. The new algorithm is given as Algorithm 2.

Algorithm 2 Expanding object set

```
$\tau \leftarrow 1$
initialize $S$, $P$
repeat forever
    $EVAL(S,P)$
    while (not termination condition) do
        $EVOLVE(P)$
        $EVAL(S,P)$
    end
    $\tau \leftarrow \tau + 1$
    add $\sigma_\tau$ to $S$
end
```

4.2 Shifting window

There are a number of reasons why we may not want to use an ever expanding set of objects to find a mapping from object space to class space. For one, this requires an ever increasing amount of computational resources, both in terms of memory and in CPU cycles. And secondly, the mapping may change over time so that obtaining $c(\sigma)$'s might prove to be an expensive operation or it is even possible that old $c(\sigma)$'s are not obtainable at all. In this case we can impose a shifting window on S limiting the number of objects to be used in the evolutionary process at any given time. The size of the shifting window will be indicated with w. The new algorithm is given in Algorithm 3.

Algorithm 3 Shifting window

```
$\tau \leftarrow 1$
initialize $S$, $P$
repeat forever
    $EVAL(S, P)$
    while (not termination condition) do
        $EVOLVE(P)$
        $EVAL(S, P)$
    end
    $\tau \leftarrow \tau + 1$
    add $\sigma_\tau$ to $S$
    if $(\tau > w)$ then remove $\sigma_{\tau-w}$ from $S$
end
```

4.3 Age

One thing we lose by employing a shifting window is the information on how well trial solutions performed on objects no longer contained in S. And the smaller w is, the greater this loss. To preserve this information in our shifting window CEA we introduce the concepts of member age and member score. The age of a member is defined as the number of population generations since the creation of that member and is denoted with P_i^{age}. The score of a member is defined as the number of correct classifications it has made since its creation and is denoted with P_i^{score}. The fitness function is now defined as:

$$FITNESS(P_i) = \frac{P_i^{score}}{P_i^{age}} \quad . \tag{9}$$

And the evaluation component becomes:

$$\begin{aligned} &EVAL(S, P) : \forall P_i \in P : \forall \sigma \in S : \\ &\quad P_i^{age} \leftarrow P_i^{age} + 1, P_i^{score} \leftarrow P_i^{score} + RESULT(\sigma, P_i) \\ &\quad \text{and compute } FITNESS(P_i) \end{aligned}$$

4.4 Two pool

One of the consequences of the new way of determining fitness is that as the age of a member increases so does its statistical reliability in approximating the *true* fitness of a member, that is, its fitness if computed using S equal to the entire object space. If, when producing offspring, the new member's score and age are set to zero, as opposed to basing them on those of its parent(s), its statistical reliability plunges and time is needed to recover some measure of reliability. In that case it is necessary to prevent the new member from participating in the evolution process until it *matures.* This can be accomplished by splitting the population into two pools, namely a child pool P^c and an adult pool P^a with $P = P^c \cup P^a$, $|P^c|$ the number of members in P^c , $|P^a|$ the number of members in P^a and *age_threshold* the age at which members are moved from P^c to P^a. The resulting algorithm is given in Algorithm 4.

Algorithm 4 Two pool

```
$\tau \leftarrow \tau + 1$
initialize $S, P^c$
repeat forever
    $EVAL(S, P)$
    while (not termination condition) do
        if ($|P^a| > 0$) $EVOLVE(P^a)$
        $EVAL(S, P)$
        $\forall P_i \in P^c$ : if ($P_i^{age} = age\_threshold$) move $P_i$ from $P^c$ to $P^a$
    end
    $\tau \leftarrow \tau + 1$
    add $\sigma_\tau$ to $S$
    if ($\tau > w$) then remove $\sigma_{\tau - w}$ from $S$
end
```

5 A new adaptive information filtering system prototype

The prototype AIF system introduced in [9] was completely rewritten incorporating the new distance measures presented in section 3 and using the two-pool CEA derived in section 4. Another change is that the weights are expressed in floating point numbers instead of integers, allowing much more gradual change during mutation. A significant improvement has been made in how the system measures its performance; in addition to tracking the lowest, average and highest fitness values, the new system also measures the actual system performance. System performance is expressed in correct classifications per document, ranging from zero for all documents classified incorrectly, to one for a perfect classification record. While the fitness values offer insight into how the CEA is doing and can, to a certain degree, be indicative of how the system is performing, system performance is by far the best basis for comparisons.

In order to accurately measure the performance of the system thousands of documents need to be classified. The $c(\sigma)$'s should to be provided via user feedback. Until the system is ready for trial deployment, however, it will be necessary to simulate this user feedback. One way this can be accomplished is by employing a test set of documents for which the $c(\sigma)$'s are known. The CEA is a special case of Algorithm 4, namely with shifting window size set to one and with a termination condition such that the inner loop is executed only once for each outer loop. The population members each consist of their score, their age, the radius parameter used by one of the *CLASSIFY* functions and a full set of weights. The system can then be described as given in Algorithm 5.

Algorithm 5 AIF two pool

```
$\tau \leftarrow 1$, initialize prototype vectors
initialize $P^c$
repeat forever
    $EVAL(\sigma_\tau, P)$
    if $(|P^a| > 0)$ $EVOLVE(P^a)$
    $\forall P_i \in P^c$: if $(P_i^{age} = \mathit{age\_threshold})$ move $P_i$ from $P^c$ to $P^a$
    $\tau \leftarrow \tau + 1$
end
```

The prototype vectors representing the category cluster centers are initialized by calculating for each the average of a certain number of trigram vectors. The initialization of the population is done by setting the scores and ages to zero, the radius to a random value within a user specified range and assigning positive random values to the weights.

There are two $CLASSIFY(\sigma, P_i)$ functions. The one determines if the distance between σ and the closest class to σ is within the maximum class radius as set in the parameter file. If so, it returns the index of that class, if not, it returns a value indicating no class was close enough. The other simply determines the class closest to σ. The distance functions used are the

Manhattan distance function $\rho'(\overline{x}, \overline{y})$ and the Euclidean distance function $\rho(\overline{x}, \overline{y})$ as derived in section 3.

There are two evolvement algorithms, one with crossover (resulting in two children produced by two selected parents) and one without crossover (resulting in one child which is a copy of the selected child). In both algorithms the generated child(ren) are mutated (see below) and the weakest adult(s) is (are) removed for the generated child(ren).

The form of crossover employed is uniform crossover, in which each gene of a child has an equal chance to come from either parent. Mutation is performed by adding with a certain probability Gaussian noise to the genes of a member. Parent selection is done by selecting fitter members with an exponentially higher probability; this causes selective pressure. If no adult gets selected by this process, the fittest adult is selected by default.

The user definable parameters for the new AIF system are as follows. For the CEA the user can specify the size of the population (positive integer), the age threshold (positive integer), the number of adults to replace after each evaluation (positive integer), the selective pressure rate (real value between 0 and 1), crossover (enabled/disabled), the chance that a gene gets mutated (real value between 0 and 1) and the amount of Gaussian noise used during mutation (real value between 0 and 1). Note that after two times the age threshold generations, the size of the child pool is the age threshold times the number of adults to replace after each evaluation, assuming the total population size is larger or equal. So, for example, if the size of the population is 100, the age threshold 10 and the number of adults to replace after each evaluation is 4, then after 20 generations the child pool will stabilize at size 40 and the adult pool at size 60. For the clustering algorithm the user can specify the distance function to be used (Manhattan or Euclidean), the number of vectors used for averaging during the initialization of the prototype vectors (positive integer) and the range of the radius values (positive real values). For each experiment the user can further specify the number of clusters and the size and number of passes for the training and the test set.

6 The Reuters-21578 text categorization test collection

The experiments conducted with the first prototype of the AIF system used Internet newsgroup articles from a number of carefully selected moderated newsgroups. This is not satisfactory for two reasons. First, while the moderation process tends to eliminate most of the personal messages, it allows a lot of meta-messages, such as announcements, the topics are often interpreted very broadly and the article contents can be relevant to multiple topics. And secondly, unless one carefully archives, indexes and makes available, the articles used in an experiment, it is not possible for other researchers to reproduce reported experimental results.

A collection of documents without the above mentioned drawbacks was desired to facilitate experimentation with the new AIF system. The construction of a large high-grade text categorization test collection is extremely time consuming, therefore we decided to use a standardized collection instead of creating one of our own. The collection we selected was the Reuters-21578 text categorization collection.

The documents in the Reuters-21578 collection appeared on the Reuters newswire in 1987. The collection is downloadable from David D. Lewis' professional home page[1]. The documents in the Reuters-21578 collection are in SGML format and tagged for the purpose of splitting into training and test sets as used in published studies concerning text classification. This was done to allow the results of different studies to be compared. For our purposes, however, a subset of the collection was needed. First of all it was required that a document be indexed with only one topic, which limited the subset to 9494 documents. And, secondly, it was required that the document be a regular text document which further limited the subset to 8654 documents. From that subset only those documents belonging to the ten most frequent topics in the subset, as listed in Table 1, were employed.

The source code for extracting the textual documents from the SGML collection file is presented in Appendix A. The extractor program scans the SGML file, checking each of the 21578 document tags to find the single topic regular text documents belonging to the topics listed in Table 1 and saves those documents as regular text files in subdirectories named for the ten topics. The source code for creating trigram frequency vector files from the extracted documents is presented in Appendix B. The trigram program treats text files as a string of characters, using a shifting window of size three to determine the trigram frequencies. Letters are handled case-insensitive and all other characters are interpreted as the space character. Any sequence of spaces is replaced by a single space. Thus the trigram alphabet consists of 27 characters, namely 'a' through 'z' and the space delimeter. The number of distinct trigrams is then $27^3 = 19683$.

We did experiments using a growing number of the selected topics in Table 1 from the Reuters-21578 collection. Our results are given in Table 2. The experiments used the Manhattan metric as distance measure. It was decided

[1] currently at `http://www.research.att.com/home/lewis`

Table 1: Subset of Reuters-21578 used in experiments

tag	topic	size
acq	Mergers/Acquisitions	2125
coffee	Coffee	114
crude	Crude Oil	355
earn	Earnings and Earnings Forecasts	3735
interest	Interest Rates	211
money-fx	Money/Foreign Exchange	259
money-supply	Money Supply	97
ship	Shipping	156
sugar	Sugar	135
trade	Trade	333

Table 2: Test set results (percentage correctly classified)

Topics	Unweighted	Average	Best	System
Coffee, trade	99.0	99.5	100	100
+ crude	93.3	98.6	100	98.7
+ money-fx	89.5	96.6	98.1	96.5
+ sugar	89.2	97.0	100	95.6
+ money-supply	83.1	93.9	100	89.7
+ ship	78.5	89.2	96.3	85.9
+ interest	77.2	88.2	93.7	84.9

to classify in closest cluster regardless of distance to that cluster. We averaged 30 document vectors in order to properly initialise the prototype vectors. For each experiment the training set was comprised of thirty document vectors for each topic and the test set of fifty document vectors for each topic (except for Money-Supply the sample was slightly smaller). The population size was 200, the age threshhold 25, the number of adults which got replaced each generation was 2, the selective pressure was 0.1, crossover was enabled, the mutation chance was 0.5, the mutation rate was 0.00001 and the training set was presented 20 times.

The first column of Table 2 lists the test set results for classifying without the use of weights. The second column lists the average adult population member score, the third column the best adult population member score and the fourth column the system score. The results show that the new matching technique presented in this report allows even unweighted trigram analysis to perform reasonably well for a small number of topics. When the number of topics increases the superiority of weighted trigram analysis is clearly demonstrated by the system scores. Preliminary results indicate that when progressively more training time is allocated as the number of topics increases, the test set results for weighted trigram analysis are greatly improved.

7 Conclusions

In this report we described a complete revision of the prototype AIF system introduced in [9] and [10]. From the results presented in section 6 we can draw a number of conclusions. First of all, the discriminating power has been significantly increased as a result of the new matching technique presented in section 3. Secondly, the combination of the new matching technique and the AIF two-pool CEA delivers greatly improved system performance. As a result of the improved system performance it is now feasible to experiment with eight and more clusters instead of only four clusters (more than four clusters caused strong degradation of performance in the old system). But while the case for generalization and scalability has been further strengthened, there is still a lot of work to be done to prove it conclusively.

Obviously a lot more experimental data is needed. A major hurdle has been the amount of computational time required to perform an experiment, as well as huge long term storage and RAM requirements. The recent move in long term storage from huge sparse trigram frequency vectors to compact trigram frequency vectors resulted in a reduction in the amount of storage space required of between 90 and 95 percent. We are now looking into doing the same for the internal representation of the trigram frequency vectors and possibly the weight vectors too, which should reduce RAM requirements comparably. It should also reduce the amount of computational time significantly allowing much larger experiments. Another area we have to concentrate on is the fine tuning of the two-pool EA. Other potential improvements to our AIF system we will investigate are support for n-grams with user definable values of n and larger alphabets. Further in the future we will be looking at more advanced clustering algorithms which will be able to add new clusters and in which each cluster would have an independent radius.

References

[1] Boyce, Bert R., Meadow, Charles T., Kraft, Donald H. (1994). Measurement in Information Science, Academic Press.

[2] Cavnar, William B. (1994). “Using An N-Gram-Based Document Representation With A Vector Processing Retrieval Model” in “Overview of the Third Text REtrieval Conference (TREC-3)”, D.K. Harman (ed.), National Institute of Standards and Technology (NIST) Special Publications 500-225, April 1995.

[3] De Heer, T. (1982). “The Application of the Concept of Homeosemy to Natural Language Information Retrieval”, Information Processing and Management **18**, No.5, pp.229–236.

[4] Jones, Karen Sparck, Willett, Peter (eds.) (1997). Readings in Information Retrieval, Morgan Kaufman, July 1997.

[5] Michalewicz, Zbigniew (1996). Genetic Algorithms + Data Structures = Evolution Programs, 3rd revised and extended edition, Springer-Verlag.

[6] Rijsbergen, C.J., van (1979). Information Retrieval, 2nd edition, Butterworths, London.

[7] Schmidt, S., and Teufel, B. (1988). “Full text retrieval based on syntactic similarities”, Information Systems, Vol. 13, No. 1, pp. 65–70.

[8] Sheth, Beered Dilip (1994). “A Learning Approach to Personalized Information Filtering”, M.Sc. thesis, Massachusetts Institute of Technology, United States of America.

[9] Tauritz, Daniel R. (1996). Adaptive Information Filtering as a means to overcome Information Overload, M.Sc. thesis, Internal Report 96–35, Department of Computer Science, Leiden University, The Netherlands. Available via:
`http://www.wi.leidenuniv.nl/MScThesis/IR96-35.html`

[10] Tauritz, Daniel R., Kok, Joost N., Sprinkhuizen-Kuyper, Ida G. (1997). “Adaptive Information Filtering using Evolutionary Computation”, Joint Conference of Information Sciences 1997, Vol.1, pp.77–80.

A Appendix A - C++ source code for document extraction

Here is a sample of reut2.sgm:

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"
OLDID="5552" NEWID="9">
<DATE>26-FEB-1987 15:17:11.20</DATE>
<TOPICS><D>earn</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;F
&#22;&#22;&#1;f0762&#31;reute
r f BC-CHAMPION-PRODUCTS-&lt;CH   02-26 0067</UNKNOWN>
<TEXT>&#2;
<TITLE>CHAMPION PRODUCTS &lt;CH> APPROVES STOCK SPLIT</TITLE>
<DATELINE>    ROCHESTER, N.Y., Feb 26 - </DATELINE><BODY>Champion
Products Inc said its board of directors approved a two-for-one
stock split of its common shares for shareholders of record as of
April 1, 1987. The company also said its board voted to recommend
to shareholders at the annual meeting April 23 an increase in the
authorized capital stock from five mln to 25 mln shares.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
```

This particular sample is converted and then saved in the file *0.art* located in subdirectory *earn* and looks like this:

```
Champion Products Inc said its
board of directors approved a two-for-one stock split of its
common shares for shareholders of record as of April 1, 1987.
    The company also said its board voted to recommend to
shareholders at the annual meeting April 23 an increase in the
authorized capital stock from five mln to 25 mln shares.
 Reuter
&#3;
```

The C++ source code of the extractor programs is as follows:

```
// Title       : Reuters collection extractor
// Author      : Daniel R. Tauritz
// Created     : 15 September 1998

#include <fstream.h>
#include <stdlib.h>
#include <string.h>

void my_itoa(int,char[]);

void main(void) {
  // Initialize
  typedef char string[30];
  string
  topic[10]={"acq","coffee","crude","earn","interest","money-fx",
             "money-supply","ship","sugar","trade"};

  unsigned topic_counter[10]={0,0,0,0,0,0,0,0,0,0};
  char line[256],label[50];
  unsigned article,counter,marker,ch,listed,loop;
  unsigned single_topic_articles=0,listed_articles=0,
           listed_normal_articles=0;
  string filemask,s;

  // Open Reuters collection data file
  ifstream datafile ("reut2.sgm");
  if (!datafile) {
    cerr << "Error! Unable to open reut2.sgm" << endl;
    exit(1);
  }

  // Read data file
  for (article=1;article<=21578;article++) {
    // Find topics line
    do {
      datafile.getline(line,255,'\n');
    } while (strncmp(line,"<TOPICS>",8));

    // Determine number of topics
```

```
counter=0;
for (ch=9;ch<=strlen(line)-9;ch++)
  if (line[ch]=='D') counter++;

// Continue processing article if single topic
if (counter==2) {
  single_topic_articles++;
  // Determine topic label
  marker=11;
  do {
    label[marker-11]=line[marker++];
  } while (line[marker]!='<');
  label[marker-11]='\0';

  // Continue processing if listed topic
  listed=0;
  for (loop=0;loop<10;loop++)
    if(strcmp(topic[loop],label)==0) listed=loop+1;

  if (listed) {
    listed_articles++;

    // Find text line
    do {
      datafile.getline(line,255,'\n');
    } while (strncmp(line,"<TEXT",5));

    // Continue processing if content type is normal
    if (line[5]=='>') {
      listed_normal_articles++;

      // Construct filename
      strcpy(filemask,topic[listed-1]);
      strcat(filemask,"\\");
      my_itoa(topic_counter[listed-1],s);
      strcat(filemask,s);
      strcat(filemask,".art");

      // Find start of body
      for (loop=0;loop<6;loop++) datafile >> s[loop];
      s[6]='\0';
```

```
          while (strcmp(s,"<BODY>")!=0) {
            for (loop=0;loop<5;loop++) s[loop]=s[loop+1];
            datafile >> s[5];
          }

          // Open file for writing
          ofstream destfile (filemask);
          if (!destfile) {
            cerr << "Error! Unable to open destination file." << endl;
            exit(1);
          }

          // Extract article to file
          for (loop=0;loop<7;loop++) datafile.get(s[loop]);
          s[7]=’\0’;
          while (strcmp(s,"</BODY>")!=0) {
            destfile << s[0];
            for (loop=0;loop<6;loop++) s[loop]=s[loop+1];
            datafile.get(s[6]);
          }

          // Close file and increase counter
          destfile.close();
          topic_counter[listed-1]++;
        }
      }
    }
  }

  // Close data file
  datafile.close();

  // Print statistics
  cout << "Single topic articles   : " << single_topic_articles << endl;
  cout << "Total listed articles : " << listed_articles << endl;
  cout << "Listed ’normal’ articles: " << listed_normal_articles << endl;
}

void reverse(char s[])
{
  char c;
  int i,j;
  for (i=0,j=strlen(s)-1;i<j;i++,j--) {
    c=s[i];
    s[i]=s[j];
    s[j]=c;
  }
}

void my_itoa(int n,char s[])
{
  int i,sign;
  if ((sign=n)<0)
    n=-n;
  i=0;
  do {
    s[i++]=(char)(n%10+’0’);
  } while ((n/=10)>0);
  if (sign<0)
    s[i++]=’-’;
  s[i]=’\0’;
  reverse(s);
}

// --- End of file ---
```

B Appendix B - C++ source code for trigram frequency vector creation

The sample text file presented in appendix A is converted and saved as file *0.vec* in subdirectory *earn* and looks like this:

```
314
219
2
315
1
323
2
339
1
364
1
375
1
377
1
...
```

> The 314 at the top of the file indicates the sum of the trigram frequencies. All the numbers after that are grouped into pairs, the first number indicating the trigram and the second the frequency. For example, the 219 was derived by $0 * 26^2 + 8 * 27^1 + 3 * 27^0$ which corresponds with the trigram *aid* and the 2 indicates that it occurred two times.

The C++ source code of the extractor programs is as follows:

```
// Title  : Trigram
// Author : Daniel R. Tauritz
// Created: 19 January 1999
//
// Input : Either a text file to be processed or a text file containing
//         the filenames of the text files to be processed (note that in
//         the latter case the filenames need to contain one single dot)
// Output: Trigram frequency vector files of the form:
//         {(total number of trigrams),(trigram index,trigram frequency),
//          (trigram index,trigram frequency),...,
//          (trigram index,trigram frequency)}

#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <math.h>

void loop(char *);
void process(char *,char *);

const unsigned num_of_trigrams = 19683; // 27^3

unsigned vector[num_of_trigrams];
char trigram[4];

int main(int argc,char *argv[]) {
  cout << "Performing trigram analysis" << endl;

  // Parse arguments
  switch (argc) {
  case 2:
    loop(argv[1]);
      break;
  case 3:
    process(argv[1],argv[2]);
      break;
  default:
    cout <<
     "Format: 'trigram inputfile outputfile' or 'trigram indexfile'" <<
     endl;
    exit(1);
  }
  return 0;
}

void loop(char *indexfilename) {
  ifstream indexfile (indexfilename);
  if (!indexfile) {
    cerr << "Error: Unable to open " << indexfilename << endl;
    exit(1);
  }
  char inputfilename[255],outputfilename[255];
  while (indexfile >> inputfilename) {
    strcpy(outputfilename,inputfilename);

    // Find dot in outputfilename (dot in filename required!)
    unsigned i=0;
    while (outputfilename[i]!=’.’) i++;

    // Replace suffix with "vec"
    outputfilename[i+1]=’v’;
    outputfilename[i+2]=’e’;
    outputfilename[i+3]=’c’;
    outputfilename[i+4]=’\0’;

    process(inputfilename,outputfilename);
  }
  return;
}

void process(char *inputfilename,char *outputfilename) {

  // Init trigram frequency vector
  for (unsigned i=0; i<num_of_trigrams;i++) vector[i]=0;

  // Open user specified file
  ifstream inputfile (inputfilename);
  if (!inputfile) {
    cerr << "Error: Unable to open specified file" << endl;
    exit(1);
  }

  // Perform trigram analysis
  unsigned index;
  unsigned total=0;
  trigram[0] = ’*’;
  trigram[1] = ’*’;
  char c;
  inputfile.get (c);
  trigram[3] = ’\0’;
```

```
do {
  c = tolower(c);
  if (c >= 'a' && c <= 'z')
    trigram[2] = c;
  else trigram[2] = '*';

  if (!((trigram[0] == '*' && trigram[1] == '*') ||
    (trigram[1] == '*' && trigram [2] == '*'))) {

    index = 0;
    for (unsigned pos=0; pos<3; pos++) {
      if (trigram[pos] == '*')
        index += (unsigned)pow(27,2-pos) * 26;
      else
        index += (unsigned)pow(27,2-pos) * (trigram[pos] - 'a');
    }
    vector[index]++;
    total++;
  }

  trigram[0] = trigram[1];
  trigram[1] = trigram[2];
} while (inputfile.get(c));

// Close inputfile
inputfile.close();

// Open output file
ofstream outputfile (outputfilename);
if (!outputfile) {
  cerr << "Error: Unable to open outputfile" << endl;
  exit(1);
}

// Write trigram frequency vector
outputfile << total << endl;
for (i=0;i<num_of_trigrams;i++) {
  if (vector[i] != 0) {
    outputfile << i << endl;
    outputfile << vector[i] << endl;
  }
  }

  // Close output file
  outputfile.close();

  return;
}

// --- End of file ---
```