Engene: A genetic algorithm classifier for content-based recommender systems that does not require continuous user feedback

John Pagonis, Adrian F. Clark

***Abstract*— We present *Engene*, a genetic algorithm based classifier which is designed for use in content-based recommender systems. Once bootstrapped *Engene* does not need any human feedback. Although it is primarily used as an on-line classifier, in this paper we present its use as a one-class document batch classifier and compare its performance against that of a *one-class k-NN* classifier.**

I. Introduction

Engene is a genetic algorithm based textual content classifier. It can operate both as a batch as well as an on-line (incremental) classifier. Our motivation in developing *Engene* is for use with a Web content-based recommender system in order to battle information overload [1].

Web *content-based recommender systems* filter Web pages and present their recommendations usually through a *digest*. Typically such digests are personalised collections of rank-ordered articles. Non collaborative filtering recommender systems infer a user's profile and use that to make suggestions about a subject or item of interest. They frequently achieve their profiling and therefore drive machine learning through the processing of *explicit* and *implicit* [2] user feedback. The reader interested in the plethora of recommendation systems may want to examine [3], [4].

Genetic algorithms have been used successfully in web content recommender systems [5], [6], [7], [8]. However, it is our view that genetic algorithms have lately been under-utilised in the fields of recommender systems and text categorisation. This view is also reflected in [9], [10] as well as by the absence of extensive coverage in recent popular text classification and related review papers [11], [3], [12]. This is unfortunate because genetic algorithms are not only good enough for filtering but also for serendipitously discovering pertinent content, which is a desired property for recommender systems.

II. Background

Traditionally in document classification applications, genetic algorithms discover how well they perform by asking the user to give feedback after each or few generations have been evolved. As a result, attempts that have involved genetic algorithms have required that a human user be part of the fitness function. An illustration in figure 1, shows how the use of genetic algorithms has moved from a closed loop process, for example in optimisation problems (as depicted in mode 1 of figure 1), to an open loop process used for text filtering applications (as depicted in mode 2 of figure 1).

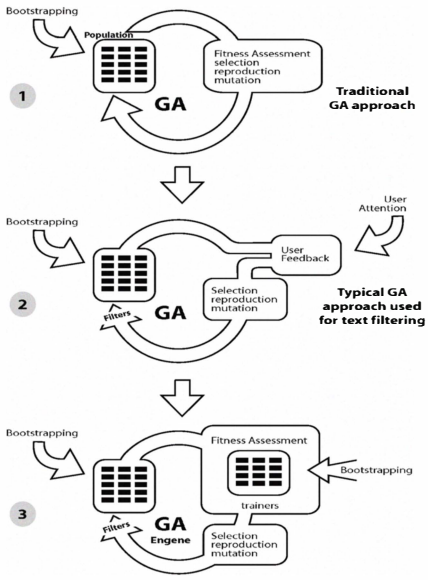

Fig. 1. GA text filtering opens the feedback loop and Engene closes it

With *Engene*, this supervision loop is closed again, taking explicit constant human feedback out of the process and therefore allowing unattended operation (as depicted in mode 3 of figure 1).

III. Engene

A popular way of representation of documents in the fields of information retrieval and filtering, is that of encoding them to multi-dimensional term vectors based on the *vector space model* [13], [14]. With this standard information retrieval method each document is processed into a vector of weighted terms. Since the *vector space model* represents a high-dimensional space in such a simple structure, it fits naturally to the *chromosome* metaphor of genetic algorithms.

Engene employs a dual-population arrangement of multi-dimensional vectors where the fitness of the evolvable information filters is assessed using a collection (also refered to as population) of trainers that never evolve, in the evolutionary

Adrian F. Clark is with the Computer Science & Electronic Engineering dept. of the *University of Essex*, Colchester, CO4 3SQ, United Kingdom
John Pagonis is with Pragmaticomm Limited, High Trees, Hillfield Road, Hemel Hempstead, Herts, HP2 4AY, United Kingdom (email: john@pagonis.org).

sense.

Having two populations of which only one is evolved has many merits, one of which is that evolution takes place unattended –without needing the user to explicitly assess the fitness of populations in every or in every few generations. Unattended evolution is achieved because what would have been a user's feedback, in *Engene*'s case, is represented by the population of trainers; which are used as input to the fitness function of the genetic algorithm.

To clarify, in this arrangement, every user interest (class) is represented by:

- A document collection that never evolves (in the evolutionary computation sense). This is referred to as *trainer* set.
- A population of information filters (the trainees) which is evolved under the direction of the trainer set. These filters are genetic algorithm individuals.

A. Bootstrapping

A drawback with this method is that during the bootstrapping process of *Engene*, the user (unless it is automated) has to source and present to the system such ensemble which is usually made of about twenty to thirty documents. Care must be taken by the user to supply documents that represent a single category of interest per ensemble. These documents are typically in HTML and have to be cleansed and processed so that stop-words and punctuation are removed. Since *Engene* currently targets only English language text, it avoids stemming [15] and applies no further dimensionality reduction to the vectors created.

One of the ways that GAs differ, from other machine learning techniques in text filtering, is that they do not inspect their training data in order to induce a classifier directly, but rather they employ the training data as the primordial genetic material to generate better filters. This genetic material comprises of the weighted terms found in the keyword vectors that represent documents in the populations evolved by the genetic algorithms. To incrementally create such genetic material (by means of mutation operators for example) would take a lot of time and make the use of genetic algorithms mostly impractical for on-line end-user systems. This is an issue which is less admitted to in the GA-based text filtering literature.

B. Encoding and term weighting

With *Engene*, the weighted multi-dimensional vector's term weights are represented by the $TFIDF$ measure. The $TFIDF$ weighting $W_{doc\,k}$ of every term k, in a document doc, is equal to the frequency of the term (TF) in the document, multiplied by the inverse document frequency of the term (IDF). In one of its standard forms the IDF is defined as the log_{10} of the ratio N/DF, where N is the number of documents in the presence of the system and DF is the frequency of the keyword within all examined documents. What the IDF factor tries to achieve is that a term is assigned a weight which is not only a measure of its importance within a document but also a measure of its uniqueness across all examined documents. Therefore the IDF smoothens out the impact of a term when this term appears across all documents. There are also other more sophisticated methods [16] for calculating $TFIDF$ but the principle is the same for all formulae used.

Since documents processed by the system have different lengths and contain variable collections of words, we decided to work with vectors of an arbitrary number of dimensions (i.e terms) and hence each of the genetic algorithm's population of individual's length is different. Document length normalisation in *Engene* is achieved by dividing the term frequency of every term of a document vector by the number of terms (TL) present in the document.

The IDF factor is typically calculated using the documents at the disposal of the system at the time of such calculation. For on-line information filtering where there is a constant flow of documents, a system can either have a pre-calculated set of IDF factors for each term based on some usually domain specific sampling of documents, or just do the calculation incrementally. Therefore, allowing an information retrieval system to discover a document vector with the most important terms.

Because of Engene's set-up and bootstrap mechanism, IDF works against it during evolution. Since *Engene* users are called to supply a small number (in two sets) of documents per subject of interest, it is only natural for a human to supply a collection of very similar documents. In-fact, such documents will most likely all contain the same terms. In which case, the weighting of these terms following the IDF weighting factor discussed above smoothens out the effect of some of its most important terms. As a consequence, it was observed during experimentation that the IDF adjustment of term weights during evolution was not beneficial. This happens because IDF was calculated for all the documents at the system's disposal which in *Engene's* case was a small population of trainers and filters. Therefore, it was discovered that if the IDF factor was altered or removed altogether *during the evolution of filters* then the weighting of terms yielded better filtering.

Initially this discovery seems awkward, but it is explained by the facts that:

1) users will supply very similar documents in terms of keywords
2) term weighting across documents is irrelevant to evolving better individuals which need to be representative of documents that are of interest to the user

Between the individuals that are evolved and the trainers that direct this evolution, weighting of terms based on their uniqueness across such ensemble is in other words moot. This is because a good ensemble will not have too dissimilar terms, otherwise individual offspring will not be representative of what is of interest. Probably due to the fact that fitness assignment in other systems is a function of explicit user feedback, *this point has never been raised before by popular literature on automatic text classification with genetic algorithms*. As a result, the $TFIDF$ weighting

of each term in a document vector can be transformed from the one in equation 1:

$$TFIDF = \frac{TF}{TL} \cdot log_{10}\left(\frac{N}{DF}\right) \quad (1)$$

to a simpler form that promotes the term frequency weight such as the one in equation 2 or the one discovered during experimentation shown in equation 3

$$TFIDF = \frac{TF}{TL} \cdot 1.0 \quad (2)$$

$$TFIDF = \frac{TF}{TL} \cdot log_{10}\left(\frac{N^2}{TF}\right) \quad (3)$$

For a small number of documents, such as the one used by *Engene* during evolution of filters, the $TFIDF$ weighting formula as seen in equation 1 will likely completely remove a term's effect by yielding to zero. This is because, the number of documents N and the frequency DF of a keyword within all examined documents will many times be equal, thus resulting in the $log_{10}(1)$, which is zero. For a large corpus, from which one seeks to discover a document with some unique terms, this is a valuable property but for the case examined herein it has the opposite effect. During experimentation we discovered that apart from applying no IDF, very good results were obtained with the weighting set as in equation 3. Since the collection of documents for the ensemble is between 20 and 30 and the term frequency TF is normalised with the size of the terms in a vector, the effects of $\frac{N^2}{TF}$ are positive.

However, when we use the generated filters to assess test documents (e.g. with a recommender) the weighting that we use in the similarity function is in the form of equation 1.

C. GA characteristics

To compare two vectors, *Engene* uses the *cosine similarity measure* between the two multi-dimensional term vectors [17], [13] using:

$$cosine\ measure = \frac{\sum_{k=1}^{m} W_{a\,k} \cdot W_{b\,k}}{\sqrt{\sum_{k=1}^{m}(W_{a\,k})^2 \cdot \sum_{k=1}^{m}(W_{b\,k})^2}}$$

The *fitness function* makes use of the cosine measure between the multi-dimensional vectors of a trainer and that of a filter. For each generation, every filter's fitness is assigned by comparing its similarity against that of a randomly selected document vector from the trainer population (see figure 2). In doing so, the trainer population directs the evolution of the trainee filter population. Consequently there is no need for continuous human feedback.

For the *selection* of individuals to mate, *Engene* uses a tournament selection algorithm where the fittest individual out of four randomly picked individuals is selected to become one of the two parents to produce offspring. This process is

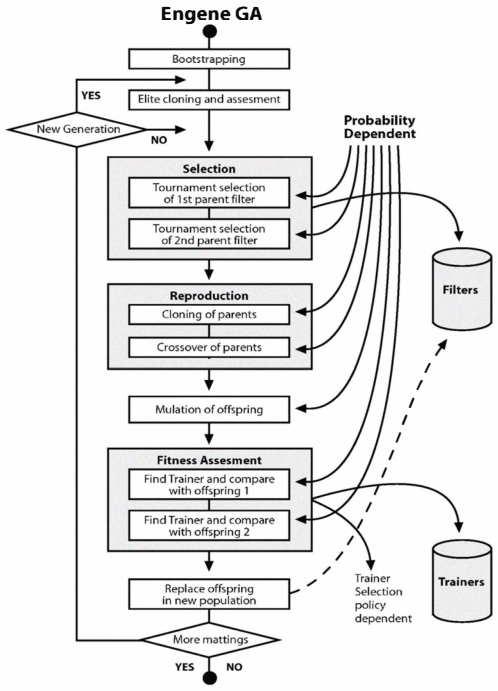

Fig. 2. Engene GA

repeated for a maximum number of mattings, equal to half the size of the population minus the percentage of the *elite*.

To keep the size of the populations manageable and in order to reduce software complexity, the genetic algorithm is designed to be *generational*; meaning that the population size is kept constant between generations. In each generation, an *elite* is chosen for immediate cloning, after which the remaining space in the new population is inhabited with individuals generated from the other operators. However, irrespective of the fact that for every generation the elite is cloned, *each of the elite's clones' fitness is re-assessed after such cloning* (due to the assessment strategy described before), thus heritage although plays its role, does not blindly guarantee an elite individual's clone a place in the future populations regardless.

For *crossover*, *Engene* uses a *random two-point crossover* operator which exchanges part of one genotype (of a filter) of an arbitrary length with part of an another. This recombination operation was implemented in two variants as depicted in figure 3. The first variant made sure that after crossover, common terms in each individual were pruned, throwing away the terms that were duplicates (irrespective of their weights), while the second variant moved any redundant terms back to the individual where they came from. Our experiments showed that the performance of the former operator was always better and thus was used instead.

The effect of the recombination operator creates filters such that on one hand are good for filtering, while on the other hand tend in time to lose some of their fitness score. This occurs because inevitably the trainers will contain

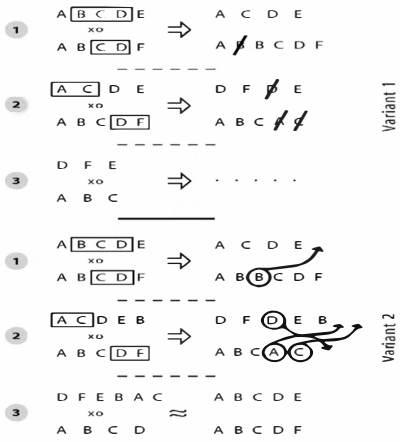

Fig. 3. Two point crossover variants

more genetic material (which means more terms each), therefore the cosine similarity measure between the vector of a filter individual and that of a trainer, will increasingly involve fewer dimensions. Effectively what happens with this crossover, is that the genetic algorithm creates a form of *dimensionality reduction* (since individuals become sorter) but without losing any genetic material from the population as a whole. Therefore population diversity is maintained while overfitting is avoided at the same time.

Finally a *mutation operator* randomly changes the value of a gene's allele in the genotype (by 20% of their weight in our experiments) while a reproduction operator simply clones individuals. Typical probability values that are used with *Engene* are 40% for crossover, 60% for cloning and 3% for mutation. Elitism is usually set at $10\% - 30\%$.

IV. Engene as one-class batch classifier

Oftentimes, classifier comparison and usage is geared toward multi-class classification. In such usage and following a training phase, classifiers are called to categorise test objects in two or more categories. Training of such classifiers is performed by supplying classifier inducers with positive and negative training data. However, when the problem at hand is to classify test objects as either "interesting" or "uninteresting", supplying representative negative examples is realistically too difficult. Therefore, in these settings classifiers have to be trained only with positive examples which is typically done as a two step process in order to source negative examples from yet unlabelled data [18], [19]. In the application of information filtering to which *Engene* is applied to, such unlabelled data are not typically present because content flows progressively through the system. Therefore *Engene* is operated for each subject of interest as a *one-class classifier that learns from positive examples only* [20]. When these classifiers are called in operation, they categorise a test object as either belonging in or out of the target category. *Engene* is one such classifier, though instead of necessarily giving a binary decision, test objects are scored and ranked instead (i.e. *Engene* is a soft classifier).

Testing how well *Engene* works as a classifier establishes its basic utility as a filter for content recommenders in general. Therefore, we tested *Engene* as a one-class classifier for its average top N precision [21]. In this measurement which is also known as the *average 11 point precision measurement*, a ranked list of test documents is evaluated at evenly spaced recall points such as the top N ranked documents, where N is $10, 20..100$. For each top N sublist therefore the *precision* is evaluated.

A. Experimental set-up

For the test, each classifier was applied as a one-class single subject filter, which was trained only on positive examples [22]. Both classifiers used the same English language documents which were not stemmed nor had gone through any dimensionality reduction. *Engene* was bootstrapped, with twenty eight documents (separated in the two ensemble sets). Similarly, the *one-class k-NN* classifier had to be trained with the same amount of data. Following the training phase, classifiers were called to rank a collection of documents.

The test corpus had 986 documents that included 29842 unique terms (after stop-word removal), which were all stored in plain text without any markup. This collection was made of technology related news articles recorded in the past four years and stored in the author's company internal so-called "FYI" system. This system records articles submitted by users that are of interest to the company and its employees. The content of these articles spans across many technology areas and interest domains, but is mostly related with mobile industry developments, products, services and technology. As such, many of the articles could have been categorised in more than one of its twenty nine classes.

For the experiments, we selected a set of twenty eight documents (from a total of 72) that represented the "Mobile Voice over IP" (MVoIP) subject of interest; this set is referred to as the *"MVoIP" set*. When experimenting with the *"MVoIP"* classification, the corresponding documents selected for training the classifiers were removed from the test set (at all times there were 958 documents of a total of 986 to be classified).

B. Results

The experiment was to classify and rank the most relevant documents to the "MVoIP" category from the "FYI" corpus and compare the performance of *Engene* against this of the *one-class k-NN* classifier.

The *one-class k-NN* classifier doesn't employ any random processes and thus is only affected, by the trainer set, the test set, the number of k trainer neighbours used and the threshold set. The threshold was not used directly, since the algorithm was made to return test object rankings. *One-class k-NN* was operated with $k = 1, 4, 8$. Typical examples for the average precision on the top N rankings (for N from 10 to 40), for the *"MVoIP"* category are shown in table I. The *recall* denominator for $TP + FN$ is adjusted to be equal to $72 - 28 = 44$; since this is the maximum number of test objects in the *"MVoIP"* category that can be retrieved.

TABLE I

ONE-CLASS K-NN TOP N AVERAGE RANKINGS, $k = 4$, "MVOIP"

Recall point (N rankings)	True Positives	Precision	Recall
10	9	90.0%	20.45%
20	15	75.0%	34.00%
30	21	70.0%	47.72%
40	24	60.0%	54.54%

As opposed to the *one-class k-NN* classifier, *Engene* makes heavy use of directed-random processes. Therefore to gauge performance, statistics had to be collected. In order to collect statistics the experiment was repeated one hundred times. The genetic algorithm evolved the ("MVoIP") filters for 100 generations each time. Then the effectiveness of *Engene* was assessed in the top 10, 20 and so on rankings. From the population of those experimental runs, the mean and standard deviation was calculated. Moreover, the experiments were repeated with various parameters. The raw results for all permutations for recall points from 10 to 40 are shown in terms of *true positives* in table II

TABLE II

ENGENE TOP N AVERAGE RANKINGS, 10-40, "MVOIP"

params.	*mean/std* 10	*mean/std* 20	*mean/std* 30	*mean/std* 40
e2-cb-es	6.82/1.36	12.72/2.41	19.32/3.03	25.2/3.68
e2-cb-ss	7.07/1.63	13.03/2.72	17.19/3.95	20.45/4.69
e2-tf-es	6.45/1.35	12.57/3.24	18.44/4.7	23.71/6.01
e2-tf-ss	5.72/2.52	9.51/4.54	12.16/6.18	14.18/6.72
e4-cb-es	6.55/1.33	12.06/2.15	18.53/2.88	24.4/3.98
e4-cb-ss	7.19/1.67	13.46/2.98	17.64/4.29	20.99/4.96
e4-tf-es	6.34/1.53	12.22/3.47	17.89/4.75	22.73/6.13
e4-tf-ss	5.69/2.72	9.37/4.64	12.16/6.14	14.33/6.31
e8-cb-es	6.81/1.18	12.78/2.02	19.37/2.71	25.33/3.26
e8-cb-ss	7.11/1.93	13.26/3.43	17.54/4.9	20.85/5.5
e8-tf-es	6.25/1.21	12.20/3.23	17.46/4.54	22.35/6.08
e8-tf-ss	6.07/2.66	9.85/4.55	13.22/6.13	15.50/6.37

The encoding on the leftmost column of these tables indicates the parameters used for each experiment (of 100 runs). The number of the best fit members from each population used for the final ranking of the test documents, is indicated by the integer next to *'e'* (e.g., *e4*). Note that this number refers to the number of filters used to rank documents and not to the elitism percentage per se that was used in the genetic algorithm during the evolution of the populations; which for the experiments summarised in table II was set to 10%.

The *'ss'* and *'es'* parameter keys indicate which *term weighting* was used during evolution. In one case *'ss'* denotes that no *IDF* factor was used, in the other case *'es'* indicates that the classical IDF ($log_{10}(\frac{N}{DF})$) was used. The key *'cb'* indicates that the trainer set and the filter set were combined in one set which was used both as a trainer and filter set. On the other hand, the *'tf'* key indicates that the two sets were kept separate and used respectively as the trainer and filter set.

From the results summarised in table II, it shows that the most competitive configuration is that of *Engene* using a set of four best-fit members (from the population of twenty

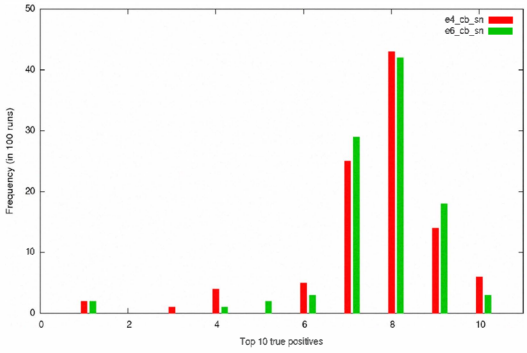

Fig. 4. Typical Engene top 10 ranking distribution

eight) to rank documents using simple TF term weighting. This is best achieved when using the combined population of trainers and filters; used as both the trainers and filters –each with the same twenty eight vectors.

For practical reasons of creating digests, in reality only the top ten rankings are of any use to a recommender. In-fact, typically the top one or two are used by a recommender, unless the digest has no other articles from other subjects of interest to present. Even so, for the top ten ranked documents the best achieved mean of *true positives* is 7.19 with a standard deviation of 1.67. In terms of average precision in the top ten ranked articles, Engene achieves 71.9% with recall of 16.34% for the "FYI" corpus.

On first sight, this classification effectiveness can be characterised as adequate but it is certainly not excellent. Although not tested on the same data set, some other genetic algorithm based efforts have reported better results, albeit using constant user feedback. Nevertheless, our system's performance is comparable to human [23] classification ability as well as to those of other classifiers. Very importantly though, we found that for every experimental run the top ranked document was never misclassified.

Furthermore, on closer inspection of the document rankings we observed an interesting property of *Engene*, this of most of its misclassifications being *near positives*. When going over the documents that were misclassified as *false positives* in the top ten lists, it was observed that those came from categories close to the required one. It appears that oftentimes documents that were ranked in the top ten and regarded as false positives, could have actually been of relevance to a user interested in the required (correct) class. This important observation has been difficult to formalise. However, this is certainly a desired property that can be used to bring serendipitous recommendations to users.

C. Improving performance

Having said that, we wanted to improve *Engene*'s performance in two aspects, this of increasing the average precision and that of the likelihood of producing high top-ten true positives (hence decreasing the standard deviation of the former average).

TABLE III
IMPROVED ENGENE TOP N AVERAGE RANKINGS, 10-50, "MVoIP"

params.	*mean/std* 10	*mean/std* 20	*mean/std* 30	*mean/std* 40
e4-cb-en	6.54/1.12	12.58/2.00	19.48/2.42	25.33/3.05
e4-cb-sn	7.56/1.59	14.58/3.00	18.56/3.89	22.26/4.67
e4-tf-en	6.33/1.16	12.54/3.24	18.22/4.64	23.65/6.09
e4-tf-sn	5.73/2.65	9.68/4.77	12.56/5.96	14.91/6.28
e6-cb-sn	7.65/1.39	14.33/2.72	18.07/3.54	21.68/3.84

In order to increase the average classification precision, experiments were conducted with the $TFIDF$ weighting adjusted as reported in equation 3 (denoted by the 'sn' key). Using this term weighting led to the *'e4'* results shown in table III, in which the improvement reported is marginally above 5%. However, more importantly this change also resulted in a much decreased standard deviation for the configuration *'e4-cb-sn'*. The distribution of the top ten true positives was further improved when we tried to increase the genetic algorithm's elitism level to 20% and we subsequently used the top six best-fit individuals to classify the "FYI" test corpus. In this case, *average precision* reached 76.5% with *recall* at 17.38%. The improvement that the *'e6-cb-sn'* configuration brings is shown in figure 4, where it is clear that almost one in two ranking runs result in eight out of ten true positives making it to the top ten document list. It is extremely important to note that these improvements shift the weight of the distribution significantly above the $\frac{6}{10}$ mark. In-fact in the *'e6-cb-sn'* configuration there is a 92% chance that *Engene* will produce between 7 and 10 true positives in the top ten list, whereas the chance of having less than six out of ten true positives is only 5%. Furthermore, in practice user experience is even better when one considers that the majority of false positives are really *near positives*, which typically have a very good chance of being relevant to the user.

V. CONCLUSIONS

In presenting *Engene* we have demonstrated how a genetic algorithm based classifier can be used successfully and without needing constant user feedback during evolution. As a result, it can be used with content recommenders that operate on implicit feedback only. We have also explained how the classical $TFIDF$ measure may be enhanced for our technique. We also believe that we have given a plausible explanation as to why genetic algorithm classifiers have been neglected recently. Moreover we argued that the presented classifier which has a classification performance comparable to humans is also beneficial because it serendipitously discovers content of interest to the user; which is a welcoming property for recommender systems.

ACKNOWLEDGMENTS

John Pagonis would like to thank Dimitris Kogias and Dimitris Papanikolaou for their support during his research as his work was supported in part by Pragmaticomm Limited.

REFERENCES

[1] J. Pagonis and M. Sinclair, "Evolving personal agent environments to reduce internet information overload: initial considerations," in *Proceedings of the IEE Colloquium on Lost in the Web: Navigation on the Internet*, ser. 1999, no. 169. London, UK: IEE, November 1999, pp. 2/1–2/10.

[2] D. Oard and J. Kim, "Implicit feedback for recommender systems," in *Proceedings of Recommender Systems Papaers of the AAAI 1998 Workshop*, 1998, pp. 81–83.

[3] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 6, pp. 734–749, June 2005.

[4] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *THE ADAPTIVE WEB: METHODS AND STRATEGIES OF WEB PERSONALIZATION. VOLUME 4321 OF LECTURE NOTES IN COMPUTER SCIENCE*. Springer-Verlag, 2007, pp. 325–341.

[5] *Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem*, London, UK, 1996.

[6] M. Balabanovic and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, pp. 66–72, 1997.

[7] *WebMate: a personal agent for browsing and searching*. ACM Press, 1998.

[8] V. Milutinovic, D. Cvetkovic, and M. J., "Genetic search based on multiple mutations," *IEEE Computer*, pp. 118–119, November 2000.

[9] S. J. Cunningham, J. Littin, and I. H. Witten, "Applications of machine learning in information retrieval," Annual Review of Information Science and Technology, Tech. Rep., 1997.

[10] I. Kushchu, "Web-based evolutionary and adaptive information retrieval," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 2, pp. 117–125, April 2005.

[11] F. Sebastiani and C. N. D. Ricerche, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, pp. 1–47, 2002.

[12] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[13] Y. C. S. Salton G., Wong A., "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[14] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrieval," *Cornell University Technical Report 87-881*, 1987.

[15] V. Hollink, J. Kamps, C. Monz, and M. De Rijke, "Monolingual document retrieval for european languages," *Information Retrieval*, vol. 7, no. 1-2, pp. 33–52, 2004.

[16] A. Singhal, "Modern information retrieval: A brief overview," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 24, no. 4, pp. 35–42, 2001.

[17] S. Gerard, *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice Hall, 1971.

[18] B. Liu, Y. D. X. Li, W. S. Lee, and P. Yu, "Building text classifiers using positive and unlabeled examples," in *Proceedings of the Third IEEE International Conference on Data Mining (ICDM-03)*, November 2003.

[19] X. Li and B. Liu, "Learning to classify text using positive and unlabeled data," in *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003, pp. 587–594.

[20] D. M. J. Tax, "One-class classification: concept-learning in the absence of counter-examples," Ph.D. dissertation, Technische Universiteit Delft, 2001.

[21] L. Larkey and W. B. Croft, "Combining classifiers in text categorization," in *SIGIR-96, 19th ACM International Conference on Research and Development of Information Retrieval*. ACM Press, 1996, pp. 289–297.

[22] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*. Springer-Verlag, 1994, pp. 3–12.

[23] C. Cyril, "Optimizing convenient online access to bibliographic databases," *Information Services and Use*, vol. 4, pp. 37–47, 1984.