Mobile Intelligent Agents for Document Classification and Retrieval: A Machine Learning Approach

Jihoon Yang, Prashant Pai, Vasant Honavar*and Les Miller
AI Research Group, Department of Computer Science
226 Atanasoff Hall, Iowa State University
Ames, IA 50011. U.S.A.
email: {yang|pdpai|honavar|lmiller}@cs.iastate.edu

Abstract

This paper describes an implementation of intelligent, customizable mobile software agents for document classification and retrieval. The mobile agents are implemented using the Voyager platform. The agents learn user's interests by interacting with the user. Results of experiments using three different approaches – TFIDF, Bayesian and DistAI (neural network classifier) – for the design of trainable document classifiers are presented. The performance of each classifier with and without feature subset selection (using genetic algorithms) was explored. Experiments with retrieval of journal paper abstracts and news articles demonstrate the feasibility of using machine learning to design mobile intelligent agents for customized information retrieval.

1 Introduction

Agents are software or hardware entities that performs a set of tasks on behalf of a user with some degree of autonomy [Russell and Norvig, 1995; Honavar, 1998]. In order to do this, an agent has to embody a certain amount of *intelligence* (e.g., the ability to choose among alternative courses of action, plan, communicate, adapt to changes in the environment, and learn from experience). An agent consists of program code, a persistent internal state, and a set of attributes (e.g., movement history, authentication keys, and so on).

Design of such mechanisms for intelligent agents has been the subject of study in artificial intelligence for over three decades. A broad range of architectures for agents with differing degrees of intelligence have been proposed in the literature. These include: reactive agents which respond reactively to changes that they perceive in their environment, deliberative agents that plan and act in a goal-directed fashion, utility-driven agents that act in ways designed to maximize a suitable utility function, learning agents which modify their behavior as a function of experience, and agents that combine different modes of behavior [Russell and Norvig, 1995; Honavar, 1998].

To be useful, an agent also needs to interact with its host system and other agents. For certain applications, it is useful for agents be able to move within heterogeneous networks of computers. This is possible only if there is a common framework for agent operations across the entire network. One approach to providing such a common framework consists of an *agent infrastructure* which is built using tools that is available for supporting agent mobility and communication as they move among different computers and networks. *Mobile* agents are such agents that can move in a computer network from host to host as needed in accomplishing their tasks. Mobile agents offer a natural extension of the *remote programming (RP)* paradigm in several interesting ways. In particular, mobile agents are generally thought to be able to act with a certain degree of autonomy. That is, the agent is able to make intelligent decisions regarding its itinerary and modify it in a dynamic fashion in response to information that becomes available as it moves from one host to another. Multiple mobile agents, when equipped with the ability to communicate with each other, can negotiate, collaborate, and compete with each other as appropriate in the pursuit of their objectives [Rosenschein and Zlotkin, 1994]. Mobile agents provide a potentially efficient framework for performing computation in a distributed fashion at sites where the relevant data is available instead of expensive shipping of large volumes of data across the network. For example, in many data mining and knowledge discovery tasks, a mobile agent can visit multiple, geographically distributed, data repositories and return with *knowledge* (e.g., in the form of a few concise rules) that captures

*Vasant Honavar's research was partially supported by the National Science Foundation (through grant IRI-9409580) and the John Deere Foundation.

the observed regularities in the data. Unlike *remote procedure calls* (RPC) which require ongoing communication through a failsafe network from the time of initiation of the task until it its completion, a major advantage of mobile agents is that ongoing interaction does not require ongoing communication [White, 1997]. Also, mobile agents can easily adjust their behavior as needed based on the status of the network. For example, in case of network failure, it can wait until the network connection is restored, or it can find alternative routes.

There is considerable ongoing research on mobile agent infrastructures. Most of them have a similar architecture consisting of at least three components: agent servers, agent interface, and agent brokers (service directory). Agent servers support basic agent migration mechanisms, authentication, and perhaps provide other services. Agent brokers provide addresses of agent servers and support mechanisms for uniquely naming agents and agent servers. Agent interface is used by application programs to create and interact with agents. The different mobile agent infrastructure proposals differ from each other in terms of detailed implementation (e.g., choice of agent transport mechanism, agent programming languages, and so on). For the experiments described in this paper, we used ObjectSpace's Voyager [1] which is a platform-independent mobile agent infrastructure that is written completely in Java. (See Section 2 for details).

The recent proliferation of computers and communication networks has made it possible for individuals around the world to potentially be able to access a wide variety of information sources on the Internet. However, effective use of these information sources (e.g., documents, articles, electronic mail messages, news, and the like), requires fairly sophisticated tools for locating, classifying, and retrieving only those items that are of interest to a given user of a group of users. For instance, a researcher might be interested in recently published papers related to his or her research from diverse sources, and want to get the interesting papers only instead of reading through all the papers in those sources. Similarly, an individual might be interested in selectively retrieving and reading news articles This presents us with a *document classification problem*. This is just one of many similar tasks that need to be automated in order for people to be able to make effective use of the emerging computing, communications, and information infrastructure.

There are three major approaches to designing intelligent agents: direct programming by the user, knowledge engineering, and machine learning. Of these, machine learning is perhaps the most practical approach for domains that require customization of agent behavior for several reasons, including the fact that it requires minimal effort on the part of users and software developers [Maes, 1997].

Document retrieval has been the subject of study for several decades [Salton, 1982]. However, work on personalized document retrieval agents is relatively new. Some examples include WebWatcher [Joachims *et al.*, 1997], Personal WebWatcher [Mladenic, 1996], Fab [Balabanovic, 1997; Balabanovic and Shoham, 1997] which learn user interests using user feedback and recommend/prefetch web pages; and software agents for mail handling and electronic news filtering [Maes, 1997].

This paper presents an approach to design of intelligent agents for customized information retrieval using machine learning. The resulting agent designs are used to construct mobile agents using the Voyager mobile agent infrastructure.

2 The Voyager Mobile Agent Platform

Several groups have designed and implemented mobile agent infrastructures. Most of them provide basic capabilities for creating, transporting, and managing mobile agents. An agent system, *Agent TCL*, is designed and implemented in [Gray *et al.*, 1996; Kotz *et al.*, 1997]. It includes network-sensing tools and a docking system for agents to move between computers. Three leading commercial mobile agent systems – General Magic's Odyssey [2], IBM's Aglets [3], and ObjectSpace's Voyager – are introduced and compared in [Kiniry and Zimmerman, 1997].

Voyager [Obj, 1997] is designed to help developers produce high impact distributed systems quickly. Voyager is implemented entirely in Java and designed to use the Java language object model. Voyager allows regular message syntax to construct remote objects, send them messages and move them between programs. The Voyager *Object Request Broker (ORB)* provides services for mobile objects and autonomous agents. It also provides services for persistence, scalable group communication, and basic directory services.

Agent platforms like Odyssey or Aglets allow developers to create an agent, program it with a set of tasks, and launch it into a network to fulfill its mission. However, they have minimal support for basic distributed computing and treats agents differently than simple objects. Aglets use sockets and Odyssey uses *Remote Method Invocation (RMI)* to move agents between machines. However, none of these platforms allow sending a regular Java message to a stationary or moving agent. As a result, it is very difficult for objects to communicate with an agent after the agent has been launched and for agents to communicate with other agents.

[1] [http://www.objectspace.com/voyager]

[2] [http://www.genmagic.com/agents]

[3] [http://www.trl.ibm.co.jp/aglets]

An attractive feature of Voyager is that it seamlessly integrates distributed computing with agent technology. An agent in the Voyager system is a special kind of object that can move independently, can continue to execute as it moves, and otherwise behaves like any other Java object. Voyager enables objects and other agents to send standard Java messages to an agent even as the agent is moving. In addition, Voyager allows us to remote-enable any Java class, even a third party library class, without modifying the class source in any way. (See [Obj, 1997] for further details of the Voyager platform).

3 The Customizable Document Retrieval Problem

A primary focus of this paper is on the design of customizable agents for information retrieval in general, and document classification and retrieval in particular.

Classification of documents necessarily has to involve some analysis of the contents of a document. In the absence of a satisfactory solution to the natural language understanding problem, most current approaches to document retrieval use a *bag of words* representation of documents [Salton, 1989]. Thus, A document is represented as a vector of weights for terms (or words) from a *vocabulary*. There are several possibilities for determining the weights: binary values can be assigned for each term to indicate its presence or absence in a document; or *term frequency* can be used to indicate the number of times that the term appears in a document; or *term frequency – inverse document frequency* can be used to measure the term frequency of a word in a document relative to the entire collection of documents [Salton, 1989].

Let d be a document. Let w_i be the ith word in d. The *term frequency* of w_i, $TF(w_i, d)$, is the number of times w_i occurs in d. The *document frequency* of w_i, $DF(w_i)$, is the number of documents in which w_i occurs at least once. The *inverse document frequency* of w_i, $IDF(w_i)$, is defined as $IDF(w_i) = log(\frac{|D|}{DF(w_i)})$, where $|D|$ is the total number of documents. Then, the *term frequency – inverse document frequency* of w_i is defined as $TF(w_i, d) \cdot IDF(w_i)$ [Salton, 1989].

Either the *term frequency* or the *term frequency – inverse document frequency* is used in the classifiers chosen in this paper. The task is to design a system that accurately classifies documents as interesting or not interesting. This is accomplished by training a document classifier using a set of pre-classified documents.

4 Classifiers

The task is to derive a system that outputs correct classifications for documents represented by vectors of weights as specified in the previous section. The following three classifiers are used in the paper: TFIDF, Bayesian and DistAI. *Term frequency* is used for the weights in Bayesian and *term frequency – inverse document frequency* is used in TFIDF and DistAI. Each classifier is described briefly in this section.

4.1 Cosine of Term Frequency and Inverse Document Frequency (TFIDF)

First, a prototype vector $\vec{c}$ is generated for every class $c \in \mathcal{C}$ where $\mathcal{C}$ is the set of all classes, by combining all feature vectors

$$\vec{c} = \sum_{d \in c} \vec{d}$$

Then, the classification is to find a prototype vector that gives the largest cosine of a document (which we want to classify) and the prototype vector itself

$$\arg\max_{c \in \mathcal{C}} cos(\vec{d}, \vec{c})$$

$$= \arg\max_{c \in \mathcal{C}} \frac{\vec{d} \cdot \vec{c}}{\| \vec{d} \| \cdot \| \vec{c} \|}$$

4.2 Naive Bayesian (Bayesian)

Here, it is assumed that a term's occurrence is independent of the other terms. We want to find a class that gives the highest conditional probability given a document d:

$$\arg\max_{c \in \mathcal{C}} P(c|d) \qquad (1)$$

By Bayes rule [Duda and Hart, 1973],

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{\sum_{c' \in \mathcal{C}} P(d|c') \cdot P(c')}$$

It is clear that

$$P(c) = \frac{|c|}{\sum_{c' \in \mathcal{C}} |c'|}$$

and

$$P(d|c) = \prod_{i=1}^{|d|} P(w_i|c)$$

in the Naive Bayesian classifier that assumes independence between terms. Also, $P(w_i|c)$ can be obtained as suggested in [Joachims, 1996]

$$P(w_i|c) = \frac{1 + TF(w_i, c)}{|F| + \sum_{w' \in |F|} TF(w', c)}$$

where $|F|$ is the size of the feature vector. Then (1) becomes

$$\arg\max_{c \in \mathcal{C}} \frac{P(c) \cdot \prod_{i=1}^{|d|} P(w_i|c)}{\sum_{c' \in \mathcal{C}} P(c') \cdot \prod_{i=1}^{|d|} P(w_i|c)}$$

$$= \arg\max_{c \in \mathcal{C}} \frac{P(c) \cdot \prod_{w_i \in F} P(w_i|c)^{TF(w_i,d)}}{\sum_{c' \in \mathcal{C}} P(c') \cdot \prod_{w_i \in F} P(w_i|c')^{TF(w_i,d)}}$$

4.3 Inter-pattern Distance-based Neural Network Classifier (DistAI)

DistAI is a simple fast constructive neural network learning algorithm for pattern classification. The key idea behind DistAI is to add hidden neurons one at a time based on a greedy strategy which ensures that the hidden neuron correctly classifies a maximal subset of training patterns belonging to a single class. Correctly classified examples can then be eliminated from further consideration. The process terminates when this process results in an empty training set (when the network correctly classifies the entire training set). When this happens, the training set becomes linearly separable in the transformed space defined by the hidden neurons. A perceptron learning algorithm (e.g., *pocket* algorithm [Gallant, 1993]) can be used to find a weight setting between the hidden and output neurons. In fact, it is possible to set the weights without going through an iterative process. It is straightforward to show that DistAI is guaranteed to converge to 100% classification accuracy on any finite training set in time that is polynomial in the number of training patterns. Experiments reported in [Yang *et al.*, 1998] show that DistAI, despite its simplicity, yields classifiers that compare quite favorably with those generated using more sophisticated (and substantially more computationally demanding) learning algorithms. This makes DistAI an attractive choice for experimenting with evolutionary approaches to feature subset selection for neural network pattern classifiers.

4.4 Feature Subset Selection

For large vocabularies, vector representation of documents results in very high-dimensional inputs to the classifier. There are many advantages to reducing the input dimension using feature or term subset selection (the process of selecting a subset of the features from a much larger set of possibly redundant, irrelevant, or misleading features) [Yang and Honavar, 1998a]. Several authors have explored the use of feature subset selection in the design of document classifiers [Koller and Sahami, 1996; 1997; Yang and Pedersen, 1997]. Genetic algorithms offer an attractive approach to feature subset selection in a variety of pattern classification applications including document classification, as demonstrated in [Yang and Honavar, 1998a]. Thus, the experiments reported in this paper explored the use of feature subset selection using genetic algorithms.

5 Implementation Details

The TFIDF, Bayesian, and DistAI classifiers were implemented for document classification with and without feature subset selection using a genetic algorithm [Yang and Honavar, 1998a; 1998b]. The classifiers were trained (and feature subset selection was performed) using a subset of documents classified by the user as interesting or not interesting.

The resulting classifiers were incorporated into mobile agents on the Voyager mobile agent platform. First, a mobile agent (**Agent1**) is generated for searching and retrieving a set of documents from a remote site that matches with the query given by the user. The query is used to retrieve documents that match the query. The agent is shipped to the remote site, and the agent retrieves matching documents and sends them to the local site. Then the agent dies. The user then classifies the retrieved documents as interesting or not interesting. This provides a dataset for training the document classification and retrieval agent using the approach outlined above. For example, a TFIDF based document classification and retrieval agent is designed using the training data and the resulting agent (**Agent2**) is sent to a remote site to retrieve *relevant* documents. Relevant documents are determined by the classifier at the remote site and returned to the local site. Then the agent dies.

6 Experiments

6.1 Datasets

Two real-world datasets were used in the experiments: *paper abstracts* and *news articles*.

- *Paper Abstracts* (**Abstracts**):
  404 paper abstracts published between 1995 and 1997 were chosen from IEEE Expert magazine, Journal of Artificial Intelligence Research and Neural Computation. Each abstract is converted into a feature vector after passing through the process of stemming [Porter, 1980] and stopping. The former generates stems of terms, and the latter drops general terms that do not convey important meaning. A query was given to retrieve paper abstracts that is related to some topics we were interested in. The query vector is compared with each document vector to select 100 paper abstracts which gave nonzero cosine values. Each selected paper abstract was given a feedback of relevance (i.e., classification) by a user to make a dataset. The feedback was given by two distinct users to generate two versions of datasets (**Abstract1** and **Abstract2**). There are 790 features and two classes (relevant or irrelevant) in the datasets.

- *news articles* (**Reuters**):
  The original Reuters dataset contains 22173 documents [4]. Instead of going through the process of stemming and stopping for the huge original data to generate usable datasets, the three

[4] The new version of Reuters dataset contains 21578 documents ([http://www.research.att.com/~lewis]).

datasets (that had been generated from the original Reuters dataset based on the topics) in [Koller and Sahami, 1997] are used in this paper:

- **Reuters1**: There are 1568 features, 939 documents and 6 classes.
- **Reuters2**: There are 435 features, 139 documents and 4 classes.
- **Reuters3**: There are 1440 features, 834 documents and 8 classes.

The datasets were used for comparing the performance of three classifiers with/without feature subset selection. **Agent1** obtains set of matching documents (**Abstrct1**) from the remote sites, and **Agent2** retrieves relevant documents from a remote site that has 118 paper abstracts.

6.2 Results

The results in Table 1 indicate that each classifier gave higher or comparable generalization accuracy with the selected subset of features than with the entire set of features.

TFIDF and DistAl outperformed Bayesian in almost all datasets (except **Abstract2** with GA-based feature subset). In particular, it is somewhat surprising as well as encouraging that the TFIDF classifier performs quite well despite its simplicity. We conjecture that TFIDF classifier works well with the document classification problem because documents are represented by a very sparse feature vectors, which produces well-separated prototype vectors for each class.

It is also worth noting that all three classifiers used approximately half the number of features (or terms) present in the original data set.

The mobile agents performed as expected: Instead of downloading all documents from the distributed databases, the agents worked on the remote databases, retrieved only a subset of relevant documents and sent them to the local site thereby minimizing the duration of the expensive network connection. Systematic experiments with distributed data sources, under a variety of network conditions and data source characteristics (e.g., percentage of relevant documents) remain to be done.

7 Summary and Discussion

Intelligent mobile agents offer an attractive paradigm for the design of modular, flexible, robust, scalable, and adaptive information systems for a variety of applications, including customized information retrieval. Machine learning appears to be the most practical approach to designing customizable software agents. This paper has presented the design of mobile agents for customized document classification and retrieval using the commercially available Voyager mobile agent infrastructure. The experiments reported here demonstrate the effectiveness of machine learning as a viable and practical approach to the design of such agents. In particular, the performance of the agents in terms of classification accuracy on novel documents not used during training is quite good, and can be improved further using automated feature selection. Analysis of the knowledge acquired by such agents can shed light on individual interests and preferences.

Some notable features of this work include the incorporation of intelligent agents into a mobile agent system to perform document classification and retrieval from distributed data collections. Some interesting and promising directions for further research include: design and implementation of multi-agent systems in which agents collaborate in retrieving and analyzing data from distributed data and knowledge sources and provide decision support services in complex real world applications; systematic studies to explore the relative performance advantages and disadvantages of alternative designs and implementations of such systems (including the mix of mobile and static agents); and issues in the design and implementation of suitable coordination and control structures for multi-agent systems.

Acknowledgements

The authors would like to thank Mehran Sahami for providing **Reuters** datasets. The authors also thank Darren Manning for installing Voyager on the department machines.

References

[Balabanovic and Shoham, 1997] M. Balabanovic and Y. Shoham. Combining content-based and collaborative recommendation. *Communications of the ACM*, March 1997.

[Balabanovic, 1997] M. Balabanovic. An adaptive web page recommendation service. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[Duda and Hart, 1973] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[Gallant, 1993] S. Gallant. *Neural Network Learning and Expert Systems*. MIT Press, Cambridge, MA, 1993.

[Gray *et al.*, 1996] R. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko. Mobile agents for mobile computing. Technical Report PCS-TR96-285, Dartmouth College, Hanover, NH, 1996.

[Honavar, 1998] V. Honavar. Intelligent agents. In J. Williams and K. Sochats, editors, *Encyclopedia of*

Table 1: Comparison of the three pattern classifiers constructed using the entire set of features against those constructed using the best subset selected using a genetic algorithm. The results represent averages based on a 10-fold crossvalidation. (The accuracy is shown in *All Features*, and both the number of selected features and the accuracy are shown in *GA-selected Feature Subsets*).

	All Features			*GA-selected Feature Subsets*					
Dataset	TFIDF	Bayesian	DistAl	TFIDF		Bayesian		DistAl	
Abstract1	89.0	70.0	89.0	401.8	83.2	396.6	81.2	402.2	89.2
Abstract2	91.0	76.0	84.0	407.4	83.2	395.6	89.8	389.8	84.0
Reuters1	94.1	73.1	91.6	771.0	96.0	771.0	89.8	766.0	90.2
Reuters2	81.5	72.3	88.5	222.6	86.8	209.2	86.3	222.4	90.3
Reuters3	98.7	75.3	96.4	719.0	97.9	697.8	91.7	721.0	96.2

Information Technology. Marcel Dekker, New York, 1998. To appear.

[Joachims *et al.*, 1997] T. Joachims, D. Freitag, and T. Mitchell. Webwatcher: A tour guide for the world wide web. In *International Joint Conference on Artificial Intelligence*, 1997.

[Joachims, 1996] T. Joachims. A probabilistic analysis of the roccihio algorithm with tfidf for text categorization. Technical Report CMU-CS-96-118, Carnegie Mellon University, Pittsburgh, PA, 1996.

[Kiniry and Zimmerman, 1997] J. Kiniry and D. Zimmerman. A hands-on look at java mobile agents. *IEEE Internet Computing*, July/August 1997.

[Koller and Sahami, 1996] D. Koller and M. Sahami. Toward optimal feature selection. In *Machine Learning: Proceedings of the Thirteenth International Conference.* Morgan Kaufmann, 1996.

[Koller and Sahami, 1997] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *International Conference on Machine Learning*, pages 170–178, 1997.

[Kotz *et al.*, 1997] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko. Agent tcl: Targeting the needs of mobile computers. *IEEE Internet Computing*, July/August 1997.

[Maes, 1997] P. Maes. Agents that reduce work and information overload. In J. Bradshaw, editor, *Software Agents.* MIT Press, Cambridge, MA, 1997.

[Mladenic, 1996] D. Mladenic. Personal webwatcher: Design and implementation. Technical report, J. Stefan Institute, Ljubljana, Slovenia, 1996.

[Obj, 1997] ObjectSpace, Inc. Voyager*: The Agent ORB for Java. Core Technology User Guide*, 1997.

[Porter, 1980] M Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[Rosenschein and Zlotkin, 1994] J. Rosenschein and G. Zlotkin. *Rules of Encounter : Designing Conventions for Automated Negotiation Among Computers.* MIT Press, Cambridge, MA, 1994.

[Russell and Norvig, 1995] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1995.

[Salton, 1982] G. Salton. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1982.

[Salton, 1989] G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, Reading, Massachusetts, 1989.

[White, 1997] J. White. Mobile agents. In J. Bradshaw, editor, *Software Agents.* MIT Press, Cambridge, MA, 1997.

[Yang and Honavar, 1998a] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. In *Feature Extraction, Construction and Selection - A Data Mining Perspective.* Kluwer Academic, NY, 1998. To appear.

[Yang and Honavar, 1998b] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. *IEEE Expert (Sepcial Issue on Feature Transformation and Subset Selection)*, 1998. To appear.

[Yang and Pedersen, 1997] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning*, pages 412–420, 1997.

[Yang *et al.*, 1998] J. Yang, R. Parekh, and V. Honavar. DistAl: An inter-pattern distance-based constructive learning algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, Anchorage, Alaska, 1998. To appear.