Concept Features in Re:Agent, an Intelligent Email Agent

Gary Boone
Georgia Institute of Technology
gboone@cc.gatech.edu
www.cc.gatech.edu/~gboone

Abstract

An important issue in the application of machine learning techniques to information management tasks is the nature of features extracted from textual information. We have created an intelligent email agent that can learn actions such as filtering, prioritizing, downloading to palmtops, and forwarding email to voicemail using automatic feature extraction. Our agent's new feature extraction approach is based on first learning concepts present within the mail, then using these concepts as features for learning actions to perform on the messages. What features should be chosen? This paper describes the concept features approach and considers two sources for learning conceptual features: groups defined by the user and groups defined by the agent's task. Additionally, features may be defined by vectorized examples or keywords. Experimental results are provided for an email sorting task.

Keywords
Electronic mail, intelligent agents, machine learning, information management, feature selection, concept features

1 Introduction

The Internet, World Wide Web, communications networks, and widespread computation and storage are rapidly creating a world in which information is ubiquitous and inexpensive. The disadvantage of access, however, is inundation. Popular media describe an alarming situation in which individuals are spending more and more time filtering information [10, 1]. The flood of irrelevant, often unwanted, data has created a need for intelligent information management systems that automate the selection, sorting, presentation, storage, and retrieval of emails, faxes, voicemail, phonepager text, web pages and other forms of information we encounter daily. With the emergence of mobile and palmtop computing, another significant need is the routing and downloading of information to pagers, cellular phones and palmtop computers.

To appear in the Second International Conference on Autonomous Agents (Agents '98) Minneapolis/St. Paul, May 10-13, 1998.

The intelligent agent has emerged as a paradigm in which programs are created as proxies for humans in interactions with computers. Agent-based approaches have been used in web browsing [18, 21], web shopping [7], newsgroup reading [13, 25], interface assistance [29, 24], information retrieval [15], and email management [19, 17, 20].

We have developed an intelligent information management agent based on numerical machine learning techniques. To validate our approach in a real-world setting, we chose email management as the test domain. Re:Agent is designed to allow email to be routed to arbitrary handlers that perform actions such as sorting, storing, deleting, or downloading to a voicepager or palmtop computer. This paper considers the problem of choosing appropriate features within the email to facilitate classification. Re:Agent learns not only the actions to perform on email, but also the features to be used in the classification task. The learning agent approach encourages users to aid the system in understanding their individual information but avoids the difficulty of end-user programming.

1.1 Intelligent Information Management

The automated management of electronic documents, web pages, newsgroup articles, email, and other electronic information is an active area of research for several fields including Artificial Intelligence, Machine Learning, and Information Retrieval. Text-based information is challenging for several reasons, including its lack of structure and constraints, its large state space, its varied content, and the importance and difficulty of representing the context in which the information occurs.

These challenges have led researchers to carefully structure and limit the information in many AI systems. For existing information types such as web pages and email, it is less likely that the structured approach is applicable. However, in some cases, there is structure; an intelligent system should use it when available. For example, although the body of email is unstructured, the sender, routing, and recipient information is explicitly labeled.

One solution is to design an agent capable of extracting useful statistical features from documents [12, 2]. Once transformed into a feature vector, learning algorithms can be trained to apply the appropriate actions. The use of a machine learning approach allows the email agent to learn from the user. By watching the user interact with documents, an agent can learn the information tasks, the user preferences and interests, and example solution techniques. Indeed, many researchers have recognized the utility of this approach. Terveen, for example, provides an agent/user collaboration interface for creating email-handling rules [28]. Krulwich's document handling agent queries the user to solicit categories for documents in which the user indicates interest [16].

In some systems, the features are fixed by the programmer; for example a particular set of words or fields may be prespecified. In our system, the agent learns the features automatically based on the task or on user suggestions about the training email's content. Because the features represent the conceptual contents in the email, we call these features "concept" features. The agent looks for these concepts in the data and then learns to perform the desired actions.

1.2 Email Filtering

Email provides an example of a rich information management domain. Email is typically short, less than a hundred lines, and contains a limited amount of structure. The body of an email is usually unstructured text, while the headers provide some tagged information. For example, an agent knows a priori the meaning of a *From* header: it defines the sender. Similarly, the *Date* header provides a known type of information. The *Subject* header is problematic. It says something about the contents of the email, but not in a fixed or prespecified way. Even for the headers with known content, the utility of their information is limited. Knowing the sender of an email is useful, but often the same sender discusses different topics. Some form of content understanding is required.

The current generation of email filtering packages, including Elm Filter [27], Ishmail [11], and Procmail [26], requires user-written rules to interpret and sort email. For example, when Procmail, which uses regular expressions, finds a match in a specified line of text, the email is forwarded, appended to a mailbox file, or piped to another program. Rule-based systems can be challenging for users due to the difficulty of not only determining the rule that achieves the desired result, but also writing it with the correct syntax.

There have been several attempts to automate the management of email. Maes and Kozierok [19] created an email agent that observed the user interacting with email. The system inferred rules by storing situation-action pairs; when it observed a situation similar to one it had seen before, it suggested the associated action. The features used to encode the context of an action were hard coded, including the message sender and receiver, subject keywords, and the message status, such as read or unread, replied to or not. The system used a confidence threshold to determine whether to make a suggestion to the user. A second threshold allowed the user to tell the agent to automatically execute the action without confirmation.

Payne and Edwards [20] considered two methods to learn to automatically classify email. They used features consisting of the words in the *From* and *Subject* fields and the top 10 most frequent words in the body. Stop-listing was used to remove common words such as *and* and *the*. Their classification learning algorithms consisted of a rule-induction method and a k-nearest neighbor method. Like Maes and Kozierok, they used trust thresholds to allow the agent act autonomously for high-confidence predictions.

Cohen [5] used a weighted vector approach similar to the one we used. He created an algorithm that learned rules based on keywords in particular fields. The emails were first structured into four attributes based on message fields: *From*, *To*, *Subject*, and *Body*, followed by the words in the corresponding part of the message, up to 100. He found that both approaches resulted in accurate email classification with only a relatively few examples. His algorithm, RIPPER, performed more accurately than the weighted vector approach, but required more time.

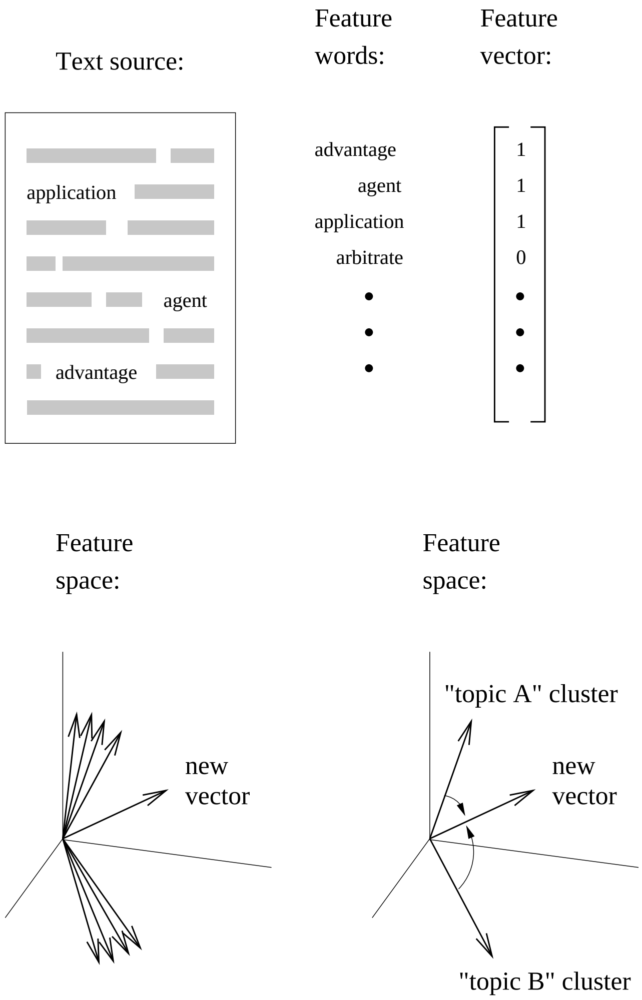

Figure 1: **Nearest Neighbor Approach.** In one common Information Retrieval approach, feature words are chosen and counted to produce a feature vector. The feature vectors of similar documents may be clustered to produce a prototype or category vector. Given query or a new document to classify, the system chooses the category nearest to the query vector. The dimension of the vector space is equal to the number of feature words, often in the thousands.

2 Concept Features Approach

Information Retrieval techniques commonly consider a set of words, usually predefined, to be features. Looking for these words in a document is called feature extraction, as shown in Figure 1. If a vector is created that counts the presence of the feature words, we can say that the document has been transformed into the feature space. Similar documents may be clustered to create prototype vectors for a given email category. New documents are then vectorized into the feature space and classified by choosing the nearest prototype.

However, instead of individual words, we can use the emails themselves to define high-level features that represent concepts present in the email. To do this, we group similar emails and combine their common information into a feature. The process of feature extraction becomes a process of determining the presence of these high-level concepts. This is the *concept feature* approach that we'll describe further in this section. As shown in Figure 2, all of the angles from the query vector to the features are used to create the feature vector.

There are several distinctions from the previously described approach. First, the features are not fixed, but defined automatically from the data or task. Second, the feature elements are statistical summaries of the training data. They may include distances to feature vectors or percentages of keywords present in the document. Third, the clusters used to create the features may be defined independently of the task. We'll consider two possible sources below. Fourth, after learning the features, a separate function approximator-based learning process learns the desired actions. Lastly, the action learner operates on a lower dimensional feature vector.

If words are used as features, the vector space has a dimensionality equivalent to the number of feature words used. This approach requires the feature words to be chosen carefully or severely limited via stop-listing, the reduction of words to a limited set of words deemed useful to categorizing [9]. Without such limits, the dimensionality will be too large to utilize machine learning algorithms, which often require low or moderate dimensions. The concept features approach reduces the dimensionality for the action learner by first learning only a relatively few high-level concept features. The word vectors, such as the *tfidf* vectors described below, can have a dimensionality equal to the number of words of the native language of the email, which is typically in the thousands. Because they are not restricted by dimensionality constraints, they may represent the email in a statistically accurate way. The action learner operates only on the distances from these vectors to the query email. Therefore, the feature space dimensionality can be small, allowing sophisticated function approximation techniques that require low dimensionality to be used.

2.1 Agent Architecture

The overall operation of our agent is based on extracting useful features from documents to create a feature vector on which machine learning algorithms operate. As shown in Figure 3, the output of the learning algorithm is an action vector which directs handlers to manipulate the document. Example handlers include delete, store, forward, autoreply, or download to pagers or hand-held devices. For the sorting experiments described below, the output of the learner specifies a folder classification for a set of test emails.

The features are defined in a simple and flexible manner, allowing the user to specify them automatically or manually as they prefer. To automatically define features, the user directs the agent to use the task training examples as the feature examples. For example, to sort all new mail from *gboone* into a *from_gary* folder, the system looks in the current *from_gary* folder to find feature examples. After learning the *from_gary* feature, it learns to sort emails with this feature into the *from_gary* folder. We call this approach *task-based* concept features.

Alternatively, the user can aid the agent by providing keywords and example messages that define *concepts* present

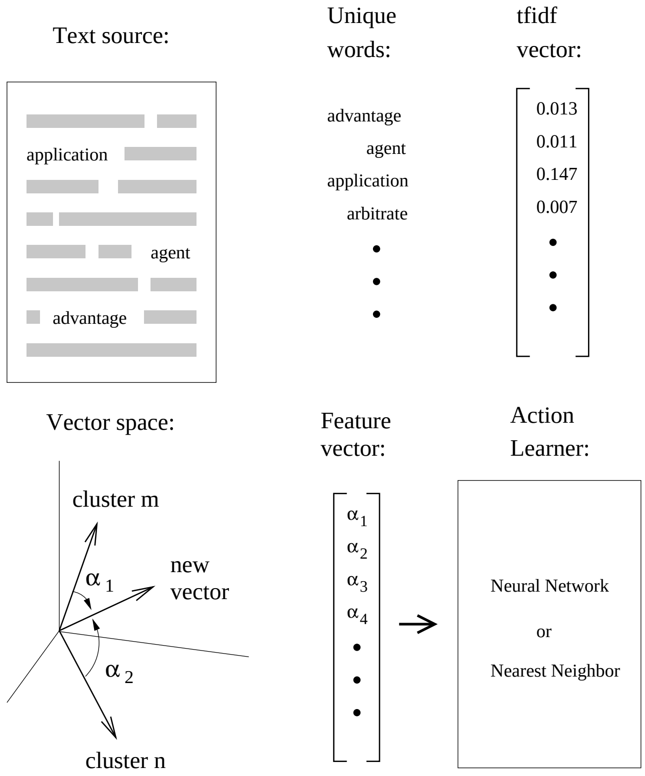

Figure 2: **Concept Features Approach.** The concept features approach uses *tfidf* weightings to create vector representations of the training emails. Similar email vectors are combined to create set of concept features. Then new emails can be cast into this concept space by creating a vector of distances to each concept vector. In this approach, a feature vector is not a list of word counts, but a list of measurements of the similarity a new email to the concept features. The size of the vector space is equal to the number of feature vectors.

in the email. The user specifies not only the task, but also instructs the agent what the email is about in a general way. For example, the task may be to learn which emails to delete and which emails to send to a voice pager. But the user has also organized the email into several categories, including *high priority*, *low priority*, *social*, and *announcements*. The agent uses these four categories to learn features, then extracts those four features from new email to decide whether to delete or forward it. We call this approach *source-based* concept features.

2.2 Feature Extraction

The purpose of feature extraction is to reduce the dimensionality of the information management problem and to improve classification performance by representing the problem in a space in which the categories are more easily separable. Unstructured text is difficult because the space of potential text strings is infinite and the meaning of words changes depending on context and word order. This section explains how keywords and example emails are used to create feature vectors.

To classify an email, Re:Agent first creates a feature vector that measures the presence of each concept feature in the new email. As shown in Figure 4, each feature contributes

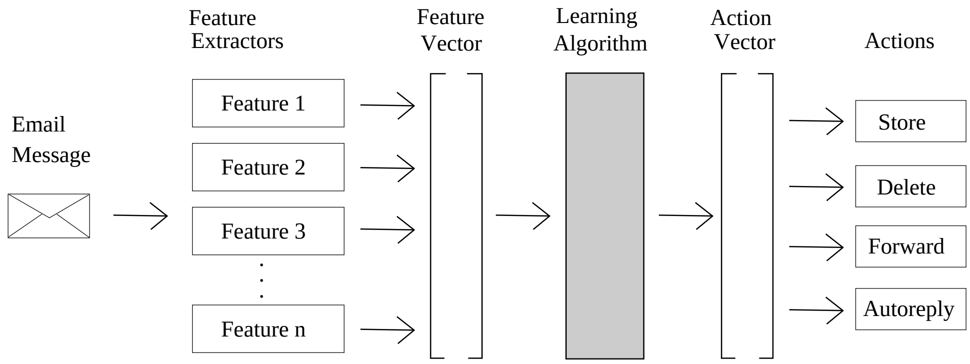

Figure 3: **Agent Architecture.** Our system extracts features from the email that can be used by a learner to determine output actions. Note that learning occurs in two places. The agent first learns what features to extract from email. It then uses the extracted features to learn what to do with the mail.

one or two elements to the feature vector. If the feature is based on example emails, the element is given by the distance from the example cluster vector to the vector of the new email. If the feature is defined by keywords, the element is given by the percentage of keywords that are present in the new email. The resulting feature vector is used by the action learning algorithm to classify the email.

If the feature uses examples, the example emails are first converted to vector space representations. Our email algorithms operate on a representation based on *tfidf* vectors commonly used in Information Retrieval [23]. An email message is first transformed into a vector of word counts where each count is divided by the number of other documents in which the word occurs. This *term frequency/inverse document frequency* approach weights each word according to how unique it is within the body of emails. This weighting captures the idea that topic words will occur frequently in a document on the given topic, but infrequently in documents about other topics. Further, because it removes any word present in every email, the technique also provides a kind of automatic stop-listing. However, note that the *tfidf* vectors can be large, often almost 15,000 words long. Our *tfidf* vector only stores words that are present in a message, even though the dimensionality of the vector is equal to the number of words in all of the email; it is thus a sparse representation.

In Information Retrieval, a *tfidf* vector is often considered a feature vector; individual words are features and the *tfidf* vector is a weighted measure of the presence of these words. In Re:Agent, *tfidf* vectors are used as a representation of email. Features are defined by clusters of emails or lists of keywords. Feature vectors, then, are lists of measurements of the presence of these clusters or keyword lists.

The *tfidf* values are calculated by first normalizing the term frequency for each term in a document,

$$\mathrm{tf} = 0.5 + 0.5\left(\frac{\mathrm{tf}_{\mathrm{raw}}}{\mathrm{tf}_{\mathrm{max}}}\right),$$

where $\mathrm{tf}_{\mathrm{raw}}$ is the number of times the term occurs in the document and $\mathrm{tf}_{\mathrm{max}}$ is the number of times the term occurs in the collection of documents. Now the terms are inversely weighted by their document frequencies,

$$\mathrm{wt} = \mathrm{tf} \times \log_2\left(\frac{N}{D_t}\right).$$

N is the total number of documents and D_t is number of documents in which the current term occurs. Note that D_t

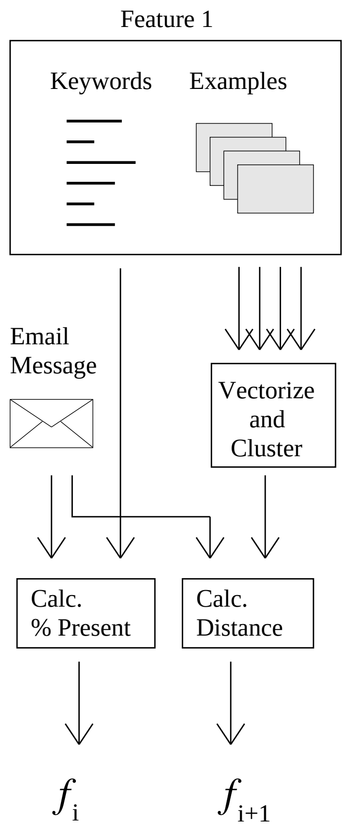

Figure 4: **Feature Extraction.** Features may contain keywords and/or example emails. To determine the feature vector on which the learner will operate, the mail is vectorized. If the feature uses keywords, the percentage of keywords present in the email is calculated and output. If there are example emails, they are vectorized and clustered; the distance between the email and the cluster center is then calculated and output. These two outputs are used as elements of the feature vector.

Source Folder	*Description*	*Number of Keywords*	*Number of Examples*	*Task Category*
papers	conference paper announcements	23	28	work
acrobot	acrobot robot discussion	35	50	work
agents	intelligent agents discussion	18	15	work
ibm	communications with IBM	29	54	work
cga	messages from Dr. Atkeson	18	34	work
jkh	messages from Dr. Hodgins	12	7	work
abowd	messages from Dr. Abowd	10	4	work
learning	discussions about machine learning	55	24	work
internships	Summer internship information	30	40	work
social	email about social events	28	254	other
spam	unsolicited commercial email	42	150	other
tara	email from family member	26	75	other
daily	other daily notes	35	475	other

Table 1: **Example Task.** For testing, we chose a filtering task using these email folders. The agent learned to correctly classify messages from these folders into the categories "work" and "other." The folders and messages are from the actual email. Two thirds of the messages were used for training and one third for testing.

is not the total number of occurrences of the term. This expression accounts for the uniqueness of words in terms of other documents, not total usage. After all of the term weights are calculated, the weighted vector is normalized by dividing each weight by the square root of the sum of the squared weights,

$$\mathrm{wt} = \frac{\mathrm{wt}}{\sqrt{\sum_{i=1}^{T} \mathrm{wt}_i^2}},$$

where T is the total number of terms in the vector. This step is the same as normalizing on all of the terms in all documents because terms that don't occur in a document are weighted to zero. Thus, we normalize all vectors into the same space.

If a feature uses examples, the example messages are vectorized, then clustered by calculating their centroid. The cluster center is stored in the feature for future use. In addition to saving calculation time during feature extraction by eliminating the need to calculate distances to every example, using the cluster center aids classification in some circumstances [22]. The similarity between vectorized messages is given by the dot product of the *tfidf* vectors. Because these vectors are normalized, this calculation gives their cosine angle; similar messages are separated by small angles.

Given an email message, the feature extraction process creates a feature representation of the message. Specifically, the angle between the vectorized message and each feature's centroid is calculated and copied into the message's feature vector. Then the percentage of each feature's keywords that are present in the message provides the remaining elements of the email's feature vector. Because the features are representations of topics, the feature vector is thus a list of beliefs, each element corresponding to a measure of the belief that a concept is present in the email. For example, if many of the keywords for a feature called *agents* are present in a message and the angle between the message's *tfidf* vector and the *agents tfidf* centroid is small, the feature vector will show that the agent believes the message contains some discussion about agents.

Feature Name	*Description*	*Number of Keywords*	*Number of Examples*
work	work-related email	36	256
other	other email	56	954

Table 2: **Task Features.** For the tests in which the task category was used to define the features, we grouped the email according to its task category. Keywords were also chosen to describe the two groups.

2.3 Action Learning with Function Approximators

After the features have been learned, the agent learns what actions to perform using the training data. To create a mapping from feature vectors to action vectors, we used a numerical function approximator. To check the sensitivity of our results to the type of function approximator, we compared the performance of two general non-linear function approximators that represent different ends of the data storage verses query speed tradeoff.

Memory-based techniques store all of the training data rather than using them to train a function approximator such as a radial basis function or neural net. When queried, memory-based systems retrieve one or more of the cases in memory and use them to construct an approximation of the output for the query point. Nearest-neighbor is the simplest memory-based technique; given a query, it returns the training output of the nearest training point. Thus, it approximates functions with piecewise constant segments.

Memory-based techniques represent one extreme of the tradeoff between rapid training and rapid querying. Their training is fast, consisting of simply storing the point. However, retrieval can be slow because upon query, a distance metric must be applied to many, possibly all, of the points in the training database.

The nearest neighbor technique was also described in the standard Information Retrieval algorithm shown in Figure 1. Note, however, that in the concept features approach, the nearest neighbor technique is applied to the feature vectors, not the *tfidf* vectors.

Neural networks are function approximators based on sums of nonlinear, typically sigmoidal, basis functions. We used a three-layer, fully-connected, feedforward network trained with backpropagation and employed crossvalidation to prevent overtraining. Neural networks create a compact representation that responds to queries quickly. However, because backpropagation is a gradient descent algorithm, they can be slow to train.

3 Experiments and Results

We considered the task of dividing several folders of email into two categories: "work" and "other." The folders contained 1,210 actual emails, listed by folder name in Table 1. We considered four experimental variations:

- How does the concept features performance compare to a standard IR algorithm?
- Should the features be defined by the task or by the source folder?
- Does the performance differ for neural network or nearest neighbor function approximators?
- Should keywords or vectors or both be used?

We used 2/3 of the 1,210 emails for training. The values presented below are the percentage of correct classifications for sorting the 404 remaining emails. Each result provides the mean and confidence interval for 17 runs. For each run, we randomly divided the emails into training and test sets. To train the neural network, the training messages were further divided in half: one half for training and the other for crossvalidation.

3.1 Information Retrieval Approach

As a comparison, we used a standard Information Retrieval technique based on *tfidf* vectors. We calculated *tfidf* vectors for each training email, then grouped them into "work" and "other" categories. Calculating the centroid for each group created a prototype vector for the each category. For testing, a *tfidf* vector was calculated for each test email and categorized according to the closest prototype vector.

3.2 Task-based Features

As described above, training can be fully automatic if the user instructs the agent to use the task actions to determine features. That is, the training examples in the classification folders are vectorized and clustered to create one feature for each output folder. We call features created with this approach *task-based features* because the clusters are calculated according to task actions.

To test this case, the test messages were vectorized and clustered to create features. Because there were two output categories, "work" and "other", there were thus two features. Then each training message was transformed into a feature vector defined by the presence of these features. These feature vectors, along with their correct categories, were used to train the neural network and nearest neighbor approximators that predicted the desired action.

3.3 Source-based Features

Our approach attempts to capture the many kinds of content that may occur in email as separate concepts. Extracting source-based features from a message transforms it into a concept space that measures the presence of the concepts.

Algorithm	*% correct*
Standard IR algorithm	90.8 ± 0.5
Concept Features (source)	97.7 ± 0.3
Concept Features (task)	97.6 ± 0.4

Table 3: **Algorithm Comparison I.** The values are the percent correctly classified, followed by their 95% confidence intervals. For the source-based concept features, the best performer used no keywords and a neural net. For the source-based concept features, the best performer used no keywords and a nearest-neighbor learning algorithm.

Because the dimensionality of the concept space is at most twice the number of features used (if every feature has both keyword and example message components), a large dimensionality reduction is achieved. For our experiments, we considered the source-based features listed in Table 2. The folder information was first used to define 13 features. Then all of the training data was transformed into feature vectors which measured the presence of these features. This new data, along with the desired "work" or "other" categories, became the training data for the neural network and nearest neighbor classifiers.

3.4 Results

The overall algorithm comparison results are shown in Table 3. The concept features approach performed significantly better than the standard Information Retrieval algorithm. Both task-based and source-based features performed equally well, providing close to 98% percent accuracy in the best case.

The results of sorting with task-based features are shown in Table 4. The use of keywords alone leads to poorer performance compared to using both keywords and example emails. Further, note that the keywords and example accuracy is less than the examples-only approach, suggesting that the use of keywords degrades performance. This is likely due to the difficulty of choosing keywords that are both complete and statistically significant. Neither the neural network nor the nearest neighbor algorithm can be said to perform better than the other overall.

When source-based features are used, as shown in Table 5, similar observations can be made. Again the example-based approach performs significantly more accurately than the versions that use only keywords. However, the use of keywords does not significantly decrease performance when both keywords and examples are used. Both the neural network and nearest neighbor algorithms are equally effective.

Function Approximator	*Keywords & Examples*	*Keywords Only*	*Examples Only*
Neural Network	82.3 ± 12.7	78.7 ± 0.0	83.4 ± 4.5
Nearest Nghbr.	96.1 ± 2.0	63.8 ± 7.7	97.7 ± 0.3

Table 4: **Learning with Task-based Features.** For task-based features, the features are defined based on the categories for the email sorting task. The values are the percent correctly classified, followed by their 95% confidence intervals.

Function Approximator	*Keywords & Examples*	*Keywords Only*	*Examples Only*
Neural Network	94.8 ± 4.2	72.7 ± 9.4	97.6 ± 0.4
Nearest Nghbr.	96.9 ± 2.3	62.2 ± 6.0	96.7 ± 0.4

Table 5: **Learning with Source-based Features.** In the source-based features approach, the features are defined by using the examples and keywords given for each source folder. The values are the percent correctly classified, followed by their 95% confidence intervals.

Comparing the task-based and source-based features performance suggests that both work equally well when keywords or keywords and examples are used. When examples alone are used, similar performance occurs for the nearest neighbor case. However, the neural network was able to perform significantly better in the source-based case. This suggests that the neural network was better able to take advantage of the larger number of features provided by the source-based approach.

To clarify the performance between source-based and task-based features, we ran an additional experiment using another 3,337 actual emails from an independent source. These emails were organized into 31 topical folders and the learning task was to sort them into 4 new categories, *research*, *interest*, *school*, and *fun*. The results of this experiment are shown in Figure 6. For both types of concept features, the nearest neighbor function approximator performed better than the neural net. For each function approximator, significantly improved performance resulted from using the topical concepts provided by the folder information.

The performance of Re:Agent suggests that accurate learning can result from well-chosen high-level features. Re:Agent allows well-chosen features to be created easily. Users need only create folders providing training data for the desired task. Additionally, they may create folders that group the training data by concepts present within the email.

4 Discussion

The concept feature approach allows Re:Agent to understand the contents of email on a conceptual level rather than a statistical word-frequency level. This approach allows the agent to more accurately spot the presence of a concept rather than the arbitrary set of words that statistically differentiates one group of messages from another. It also allows the user to instruct the agent on the contents of email in an intuitive way. Our results suggest that accurate classification can be achieved with both task-based and source-based features and without using keywords. Further, the concept features approach is robust with respect to the choice of learning algorithm.

Algorithm	*Feature Type*	*Function Approx.*	*% correct*
Standard IR			72.1 ± 0.2
Concept Features	task	neural net.	76.8 ± 0.6
		nearest nbr.	80.4 ± 0.3
Concept Features	source	neural net.	80.5 ± 0.8
		nearest nbr.	87.0 ± 0.3

Table 6: **Algorithm Comparison II.** In a second experiment with 3,337 emails, we compared the performance for features based on examples only. The means and 95%confidence intervals were calculated after 26 trials.

It may seem that there is an immense number of possible topics required to describe the contents of a user's email. However, it is not necessary to create a feature for every possible topic or even every topic present in the user's email. The agent requires only enough features to allow the accurate classification of the email. Nor does it need to parse or interpret the concepts; it only needs to recognize their presence. Although we wish to provide the agent with some ability to extract the concepts in email, we do not need complete natural language understanding of the text. The conceptual features approach represents a balance between statistical classification and complete understanding.

It may be suggested that the agent should be able to infer conceptual features automatically by choosing features which best predict the training data. Several researchers have considered this approach [3, 4, 8, 14]. For example, Latent Semantic Analysis uses singular value decomposition to find a set of orthogonal factors which best represents the semantic structure of the documents [6]. However, because any database of text examples will contain a small number of examples for each of a large number of topics, automatic feature extraction will generate many incorrect feature words [14]. The concept features approach allows the user to guide the learning process by suggesting actual conceptual features present in the text. The construction of these features is natural to users with sorted email. Future work will consider direct comparisons of these techniques.

It is unlikely that programmers can anticipate all of the information needs of the user, nor code a general-enough solution to work effectively for all users. Therefore, to manage a user's information, end-user programming and agent learning is required. We believe that allowing the user to guide the learning agent is the appropriate level of interaction for programming agents. We find that it is easier to list keywords or identify examples than to write rules.

5 Conclusion

Recognizing that similar types of information occur often in an individual's email, we have created an agent that can recognize the topics present in email. Re:Agent operates by using examples and keywords provided by the user to define concept-level features. It then uses these features to learn appropriate actions for the email. We believe that the concept features approach provides a convenient mechanism for users to explain the topics present in their email that are useful for performing the desired task. In contrast to rule-based systems which require users to write rules, Re:Agent users are required only to place example messages in folders corresponding to the desired actions. Re:Agent learns the concepts and decision policies from these folders or can utilize additional keyword and concept suggestions.

If agents are to help us manage the deluge of information that bombards us electronically, they must be given sufficient intelligence to learn from us and to collaborate with us in a way that is natural to our methods of communication. Re:Agent represents a step in that direction by learning classification features based on the topics in a user's documents.

References

[1] Managing corporate communications in the information age. http://www.pitneybowes.com/pbi/whatsnew/releases/communication_options.htm. Institute for the Future (IFTF), the Gallup Organization and San Jose State University, commissioned by Pitney Bowes Inc.

[2] Marko Balabanović and Yoav Shoham. Learning information retrieval agents: Experiments with automated web browsing. In *AAAI-95 Spring Symposium on Information Gathering from Heterogenous, Distributed Environments*, 1995.

[3] Suzanna Becker and Mark Plumbley. Unsupervised neural network learning procedures for feature extraction and classification. *Journal of Applied Intelligence*, 6:1–21, 1996.

[4] Chris Clack, Jonny Farrington, Peter Lidwell, and Tina Yu. Autonomous document classification for business. In *Proceedings of the 1997 International Conference on Autonomous Agents*, February 1997.

[5] William W. Cohen. Learning rules that classify e-mail. In *Proceeding of the AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[6] Scott Deerwester, Susan T. Dumais, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[7] B. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the 1997 International Conference on Autonomous Agents*, February 1997.

[8] David J. Finton and Yu Hen Hu. Importance-based feature extraction for reinforcement learning. In et al. Thomas Petsche, editor, *Computational Learning Theory and Natural Learning Systems*, volume 3. MIT Press, 1994.

[9] William B. Frakes and Ricardo Baeza-Yates. *Information Retreival: Data Structures and Algorithms*. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[10] S. C. Gwynne and John F. Dickerson. Lost in the e-mail. Time Magazine, April 1997. pp. 88–90.

[11] Jonathan Helfman and Charles Isbell. Ishmail: Immediate identification of important information. Technical report, AT&T Labs, 1995.

[12] David Hull, Jan Pedersen, and Hinrich Schüetze. Document routing as statistical classification. In *AAAI Spring Symposium on Machine Learning in Information Access*, Palo Alto, CA, March 1996.

[13] Tomonari Kamba, Krishna Bharat, and Michael Albers. The krakatoa chronicle: An interactive, personalized newspaper on the web. In *Proceedings of the Fourth International World-Wide Web Conference*, Boston, MA, December 1995.

[14] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In Lorenza Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*. Morgan Kaufmann, 1996.

[15] Daphne Koller and Yoav Shoham. Information agents: A new challenge for ai. *IEEE Expert*, pages 8–10, June 1996.

[16] Bruce Krulwich. Learning document category descriptions through the extraction of semantically significant phrases. In *Proceedings of the IJCAI Workshop on Data Engineering for Inductive Learning*, 1995.

[17] Yazdi Lashkari, Max Metral, and Pattie Maes. Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1, Seattle, WA, August 1994. AAAI Press.

[18] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, August 1995.

[19] P. Maes and R. Kozierok. Learning interface agents. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 459–465, Washington, D.C., July 1993. MIT Press.

[20] Terry R. Payne and Peter Edwards. Learn: An investigation of learning issues in a mail agent interface. *Applied Artificial Intelligence*, 11:1–32, 1997.

[21] Michael Pazzani, Daniel Billsus, and Jack Muramatsu. Syskill & webert: Identifying interesting web sites. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, August 1996. American Association for Artificial Intelligence.

[22] Gerard Salton. *Automatic Text Processing*. Addison-Wesley, Reading, MA, 1989.

[23] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical report, Cornell University, Department of Computer Science, 1987.

[24] Jeffrey C. Schlimmer and Leonard A. Hermens. Software agents: Completing patterns and constructing user interfaces. *Journal of Artificial Intelligence Research*, 1:61–89, 1993.

[25] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In *Ninth Conference on Artificial Intelligence for Applications*, pages 345–352, Orlanda, FL, March 1993. IEEE Computer Society Press.

[26] Germany Stephen R. van den Berg at RWTH-Aachen. procmail—autonomous mail processor.

[27] D. Taylor. Mail filtering on the fly: The elm filter program. Technical Report STL-88-4, Hewlitt-Packard Laboratories, Palo Alto, CA, February 1988.

[28] Loren G. Terveen and La Tondra Murray. Helping users program their personal agents. In *Proceedings of the 1996 Conference on Human Factors in Computing Systems (CHI '96)*, pages 355–361, Vancouver, April 1996.

[29] Andy Wood, Anind Dey, and Gregory D. Abowd. Cyberdesk: Automated integration of desktop and network services. In *Proceedings of the 1997 Conference on Human Factors in Computing Systems (CHI '97)*, 1997.