

Learning from Message Pairs for Automatic Email Answering^{*}

Steffen Bickel and Tobias Scheffer

Humboldt-Universität zu Berlin, School of Computer Science
Unter den Linden 6, 10099 Berlin, Germany
{bickel, scheffer}@informatik.hu-berlin.de

Abstract. We consider the problem of learning a mapping from question to answer messages. The training data for this problem consist of pairs of messages that have been received and sent in the past. We formulate the problem setting, discuss appropriate performance metrics, develop a solution and describe two baseline methods for comparison. We present a case study based on emails received and answered by the service center of a large online store.

1 Introduction

The email domain is an interesting, commercially relevant, and therefore very popular field of machine learning applications. Several different problem settings with distinct levels of user benefit and required user interaction have been identified. Recognition of spam emails has been studied intensely (*e.g.*, [4, 16, 6]). Similarly frequently, the problem of sorting emails into folders, prioritizing and forwarding emails has been investigated (*e.g.*, [2, 7]). All of these problems fit into the supervised learning paradigm; class labels are obtained from the folder in which the emails are found or, in case of personalized spam filtering, from additional user interaction.

By contrast, the problem of *answering* emails is not well subsumed by the classification framework. The available training data consist of pairs of inbound and corresponding outbound emails – typically, very large amounts are stored on corporate servers. In its full generality, the problem is clearly AI-complete. However, in specific environments – such as call centers – we may expect to find sufficiently strong regularities for machine learning algorithms to be applicable.

Our paper makes several contributions. We motivate and formulate the message answering problem setting. We discuss two baseline solutions: an instance-based method which retrieves the answer to the most similar inbound training message and a classification-based solution. The latter requires a user to additionally label messages; while this approach is less practical, it provides us with an approximate upper bound of how accurate we can possibly learn from just the message pairs. We then develop a solution to the message answering problem

^{*} Appearing in Proceedings of the European Conference on Machine Learning, Pisa, Italy, 2004. © Springer-Verlag.

that does not require additional user interaction. Finally, we present a case-study and evaluate the discussed methods using emails that have been received and answered in the service center of a large online store.

The rest of our paper is organized as follows. We review related work in Section 2. We formulate the problem setting and appropriate performance metrics in Section 3. In Section 4, we discuss “baseline” methods; in Section 5, we develop our solution that learns from message pairs and present a case study on these methods in Section 6. Section 7 concludes.

2 Related Work

The problem of filtering spam emails has been studied frequently. Methods based on the naive Bayes algorithm have been explored [12, 16, 10] and compared to keyword based approaches [1], rule induction [4, 14] and Support Vector Machines [6]. Machine learning of personalized spam classifiers requires the user to manually label emails as spam; for instance, the Netscape email client has been equipped with “this is spam” and “this is not spam” buttons.

Furthermore, the problem of filing emails into a list of folders has been studied. Filtering rules [11] that can be defined in email clients such as Outlook have been found to be unpopular [7]. On the other hand, training data for the email filing problem (previously filed emails) can be obtained without additional user interaction. Several methods and email assistance systems have been presented, including Re:agent [2], MailCat [18], iFile which is based on naive Bayes [15], Support Vector Machines and unigram language models [3], filtering rules [13], co-trained Support Vector Machines [9], and systems which use first-order representations and methods of inductive logic programming [19, 5].

The benefit of automatic email filing is large for users who spend substantial time sorting their emails. Addressing the problem of *email answering* is clearly more difficult, but promises a potentially much greater benefit. A system and a case study has been discussed [17], in which a classifier is used to map inbound emails to a set of manually defined standard answers. In order to train the system, standard answers have to be defined, and example inbound emails which can be answered by each of the standard answers have to be selected as training instances for the classifier. This rather high level of additional user interaction reduces the user benefit and possibly the acceptance of the system.

Our problem setting differs substantially from the *question answering* problem (*e.g.*, [8]). In our setting, many message pairs are available for training, and a large fraction of all inbound messages can be answered with a modest number of distinct answers.

3 Problem Setting

A *message answering problem* is constituted by a distribution $p(x, y)$ over *inbound messages* $x \in X$ and *outbound messages* $y \in Y$, and a binary *acceptance function*, $accept : (x, y) \mapsto \{0, 1\}$, that decides whether an answer is acceptable.

Neither $p(x, y)$ nor $accept$ are known or available to the learning algorithm. X and Y are typically the set of all strings.

The *training data* are a sequence $M = \langle (x_1, y_1), \dots, (x_m, y_m) \rangle$ of message pairs, governed by $p(x, y)$. The data contain some information about $accept$: for each message pair $(x_j, y_j) \in M$, $accept(x_j, y_j) = 1$. A *message answering hypothesis* is a mapping f that accepts an inbound message x and produces either an outbound message y , or the special symbol \perp , indicating that f is unable to answer x .

What is the appropriate performance measure for this problem? For an inbound message x , an acceptable answer $accept(x, f(x)) = 1$ is the most desirable, a wrong answer $accept(x, f(x)) = 0$ the least desirable outcome; rejection of a message $f(x) = \perp$ denotes that the message has to be answered manually.

Among the answers that are sent, we would like to have a large fraction of acceptable answers. At the same time, we desire a large percentage of the incoming emails answered automatically. We can estimate the *precision* by counting, for a given test collection, the number of acceptable answers over all answered messages (Equation 1); we estimate the *recall* as the number of acceptably answered messages over all messages (Equation 2).

$$\widehat{Precision}(f) = \frac{\text{acceptably answered messages}}{\text{answered messages } (f(x) \neq \perp)} \quad (1)$$

$$\widehat{Recall}(f) = \frac{\text{acceptably answered messages}}{\text{all messages}} \quad (2)$$

Message answering methods – such as the algorithm presented in Section 5 – will typically determine a value that quantifies their confidence. By thresholding this confidence measure, we obtain a *precision-recall curve* that characterizes a message answering method.

The *accept* function models a user’s decision about whether $f(x)$ is an appropriate answer to x . How can we estimate the performance of a message answering system without having a human “in the loop”? We manually divide a test collection into partitions $\mathbb{S} = \{S_1, S_2, \dots, S_n\}$, such that each partition $S_i = \langle (x_1, y_1), \dots, (x_{m_i}, y_{m_i}) \rangle$ contains only semantically equivalent answers. Answers that cannot be grouped into sufficiently large equivalence classes are collected in S_n , a special partition for “miscellaneous” answers. In order to evaluate the message answering methods described in this paper we define *accept* as follows. Given an example $(x_j, y_j) \in M$, the conjectured answer $f(x_j)$ is accepted – $accept(x_j, f(x_j)) = 1$ – if both y_j and $f(x_j)$ occur in the same partition S_i , and S_i is not the “miscellaneous” group S_n . Otherwise, $accept(x_j, f(x_j)) = 0$.

4 Baseline Methods

In this section we describe two baseline methods: an instance-based method that serves as a lower bound and a supervised classification method that receives additional information and can serve as an approximation to an upper bound for the performance achievable for the answering approach presented in Sec. 5.

4.1 Instance-Based Method

The learning phase of the instance-based answering method consists of just storing all training message pairs in M . Finding the appropriate answer to a new question x means finding the most similar question x_j in the training data. The corresponding answer y_j is the answer for x . According to this procedure, $f(x)$ is defined in Equation 3.

$$f_\theta(x) = \begin{cases} y_j | \operatorname{argmax}_{(x_j, y_j) \in M} \operatorname{sim}(x, x_j) & \text{if } \operatorname{sim}(x, x_j) > \theta \\ \perp & \text{otherwise} \end{cases} \quad (3)$$

The parameter θ is a measure for the desired level of confidence. We use a tfidf vector space representation for the messages. The similarity measure can be defined as $\operatorname{sim}(x_1, x_2) = \cos(x_1, x_2) = \frac{x_1 \cdot x_2}{\|x_1\| \cdot \|x_2\|}$.

After retrieving the answer y_j to the most similar training message x_j , an adaptation step that mainly involves generating a proper salutation line has to be executed. We exclude this rather technical issue from our discussion.

4.2 Supervised Classification Method

In order to find an approximation to an upper bound for the answering performance that we can achieve by an answer clustering approach as described in Section 5, we extend our training data by additional information. We group all answers into semantically equivalent classes. Mapping inbound messages to one of several equivalence classes is then essentially a classification problem. Note, however, that in an application scenario the definition of equivalence classes and partitioning of answers into those classes requires unacceptably high user effort.

The collection of training message pairs is manually partitioned into $\mathbb{S} = \{S_1, S_2, \dots, S_n\}$, where each $S_i = \langle (x_1, y_1), \dots, (x_{m_i}, y_{m_i}) \rangle$ contains only pairs such that there is a single outbound message y_i^* that is acceptable for all inbound messages in S_i : $(x_j, y_j) \in S_i \Rightarrow \operatorname{accept}(x_j, y_i^*) = 1$. Besides partitioning the messages, we require the user to formulate these answer templates y_i^* . S_n contains miscellaneous pairs that do not form sufficiently large groups of equivalent answers. We would now like to find a classifier which maps a message x to an equivalence class of answers $f(x) = y_i^*$.

We transform \mathbb{S} into a training set for the corresponding classification problem. The positive examples for class i consist of the sequence of inbound messages in $S_i = \langle (x_1, y_1) \dots, (x_{m_i}, y_{m_i}) \rangle$: $Pos_i = \langle x_1, \dots, x_{m_i} \rangle$. Since no two distinct equivalence classes are equivalent, the messages in Pos_i are negative examples for all other classes $Pos_{j \neq i}$.

We have now transformed the message answering problem into a classification problem. Now, we will describe how we solve this classification problem using Support Vector Machines. We first have to reduce the multi-class learning problem to a set of binary learning problems. In our application, the classes are very unevenly distributed. Therefore, we found both, the one-against-all and one-against-one approach to perform unsatisfactorily. We use a probabilistic one-against-all approach that explicitly considers the prior probabilities.

For each equivalence class i , we learn a decision function $g_i(x)$ using the inbound messages Pos_i as positive, and all $Pos_{j \neq i}$ as negative examples. Now we assume that the decision function values are governed by normal distributions: $P(g_i(x)|i) = N[\mu_i, \sigma_i^2]$. We estimate μ_i , σ_i , $\bar{\mu}_i$, and $\bar{\sigma}_i^2$ from the training data: we learn a decision function $g_i(x)$ and estimate mean value μ and variance σ_i^2 of the positive and $\bar{\mu}_i^2$ and $\bar{\sigma}_i^2$ of the negative examples. The prior probabilities $P(i)$ are estimated by counting the relative size of the partitions S_i . In the next step we calculate the posteriors $P_i(i|g_i(x))$ by applying Bayes' equation as shown in Equation 4.

$$P_i(i|g_i(x)) = \frac{N[\mu_i, \sigma_i^2](g_i(x))P(i)}{N[\mu_i, \sigma_i^2](g_i(x))P(i) + N[\bar{\mu}_i, \bar{\sigma}_i^2](g_i(x))P(\bar{i})} \quad (4)$$

We have several one-against-all classifiers that independently estimate $P_i(i|g_i(x))$. Variance of these estimators can lead to posteriors which do not exactly sum up to 1; we therefore normalize the posterior estimates in Equation 5.

$$P(i|x) = \frac{P_i(i|g_i(x))}{\sum_j P_j(j|g_j(x))} \quad (5)$$

A new question x is answered as follows. The collection of decision functions is evaluated, Equations 4 and 5 yield an estimate of the posterior $P(i|x)$. We select $f(x)$ to be the manually selected template for class i , y_i^* , where $i = \operatorname{argmax}_i P(i|x)$. Once again, the answer has to be adapted by formulating an appropriate salutation line. This leads to the definition of $f(x)$ in Equation 6. The found answer template y_i^* is returned as $f_\theta(x)$, if $P(i|x)$ exceeds a given threshold θ and if the equivalence class S_i is not the "miscellaneous" class S_n ; otherwise, $f_\theta(x) = \perp$.

$$f_\theta(x) = \begin{cases} y_i^* | \operatorname{argmax}_i P(i|x) & \text{if } P(i|x) > \theta, i \neq n \\ \perp & \text{otherwise} \end{cases} \quad (6)$$

5 Learning from Message Pairs

The answering method described in Section 4.2 is not suited for real applications because a user is forced to manually partition hundreds of messages into equivalence classes before an answering system can start to support the task of answering emails. This is prohibitive in practice. The method described in Section 4.1 does not need this preparatory work but uses only a limited amount of the available information; each conjectured answer is only based on one single training instance (one message pair).

To overcome these limitations, we will now discuss a method that generates a message answering hypothesis from pairs of training messages. The key idea is that we replace the manual partitioning of answer messages by a clustering step. On the answers of the training message pairs, a clustering algorithm creates a partitioning $\mathbb{C} = \{C_1, C_2, \dots, C_r\}$, where the $C_i = \langle (x_1, y_1), \dots, (x_{m_i}, y_{m_i}) \rangle$ are message pairs whose answers y_j lie in the same cluster i .

Table 1. Algorithms for learning from message pairs and answering new questions.

Learning from message pairs.

Input: A sequence $M = \langle (x_1, y_1), \dots, (x_m, y_m) \rangle$ of message pairs, variance threshold σ_τ^2 , pruning parameter π .

1. Recursively cluster answers y_j for all $(x_j, y_j) \in M$ into clusters C_1, C_2, \dots, C_{r-1} . End the recursion when cluster variance lies below σ_τ^2 .
2. Prune all clusters with less than π elements. C_r is the sequence of pairs whose clusters have been pruned.
3. For each cluster $C_i \in \{C_1, C_2, \dots, C_{r-1}\}$, find the answer template y_i^* which is closest to the centroid according to Equation 7.
4. For all clusters $C_i = \langle (x_1, y_1) \dots, (x_{m_i}, y_{m_i}) \rangle$, let $Pos_i = \langle x_1, \dots, x_{m_i} \rangle$ be the positive examples.
5. For all clusters C_i , $1 \leq i \leq r$:
 - (a) Train a Support Vector Machine with resulting decision function g_i using Pos_i as positive, and all other $Pos_{j \neq i}$ as negative examples.
 - (b) Estimate mean value and variance $\mu_i, \sigma_i^2, \bar{\mu}_i, \bar{\sigma}_i^2$ of the decision function values of positive and negative examples.
6. Return answering hypothesis f according to Equation 8.

Answering new questions.

Input: New inbound message $x \in X$, message answering hypothesis f , confidence threshold θ .

1. Calculate SVM-decision function values $g_i(x)$ for $i = 1, \dots, r$.
 2. Assume Gaussian likelihoods and estimate $P(i|x)$ according to Equations 4 and 5. Parameters $\mu_i, \sigma_i^2, \bar{\mu}_i$, and $\bar{\sigma}_i^2$ are part of the answering hypothesis f .
 3. If $\operatorname{argmax}_i P(i|x) = r$ (the ‘‘miscellaneous’’ cluster is most likely) or $\max_i P(i|x) \leq \theta$, then return \perp .
 4. Otherwise, return y_i^* with $i = \operatorname{argmax}_i P(i|x)$ (according to Equation 8).
-

For each cluster i , we construct an answer template y_i^* . Because we do not want to require a user to manually define a template as in Section 4.2, we select y_i^* to be the closest answer to the centroid of the cluster C_i (Equation 7).

$$y_i^* = \operatorname{argmax}_{y: (x,y) \in C_i} \operatorname{sim} \left(y, \frac{1}{|C_i|} \sum_{(x_j, y_j) \in C_i} y_j \right) \quad (7)$$

As clustering algorithm, we use EM with mixtures of two Gaussian models recursively. In each recursive step the data are split into two clusters until a predefined cluster variance threshold is reached or a cluster with one element remains. We now prune all clusters which do not have sufficiently many examples; C_r collects message pairs of pruned clusters. In the next step we train r binary linear SVM classifiers on the question messages according to the procedure described in Section 4.2.

A new inbound message x is answered as follows. After classifying x into a partition $C_i \in \mathbb{C}$ analogously to Section 4.2, we return the corresponding answer

template y_i^* , where i maximizes the posterior estimate $P(i|x)$. Equation 8 shows the resulting definition of $f_\theta(x)$ which is analogous to Equation 6. $P(i|x)$ is computed analogously to Equations 4 and 5. The resulting algorithm is shown in Table 1.

$$f_\theta(x) = \begin{cases} y_i^* | \operatorname{argmax}_i P(i|x) & \text{if } P(i|x) > \theta, i \neq r \\ \perp & \text{otherwise} \end{cases} \quad (8)$$

6 Case Study

The study is divided into three subsections. We first examine properties of the used data set. Secondly, we compare our method for learning from message pairs with the baseline methods; and finally, we analyze the dependency between answering performance and the parameters of the clustering algorithm.

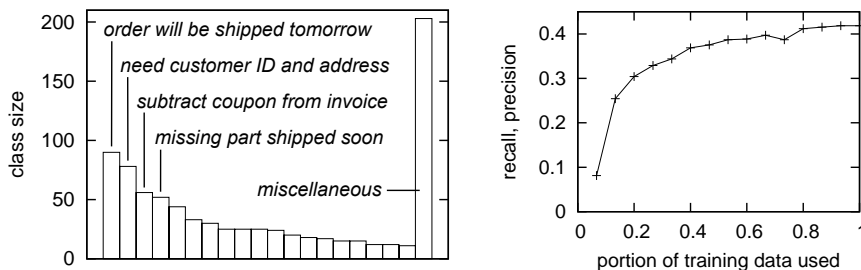


Fig. 1. Distribution of class sizes with descriptions for largest classes (left); learning curve for precision/recall of the supervised classification problem (right).

6.1 Data Set Properties

The data set used in the study was provided by a large online store. A total of 805 question-answer pairs represent the emails answered by one customer service employee within one month. The questions mostly deal with problems regarding late delivery, incomplete shipment, and defective products. In order to evaluate the performance of the message answering methods via the *accept* function, and to obtain the required partitioning for the supervised classification method, we manually group the answers into 19 semantically equivalent classes.

Figure 1 (left) shows the distribution of class sizes for the data set. We choose ten as the minimum size of a class. All messages in smaller groups fall into the “miscellaneous” group that contains 203 messages and is shown as the rightmost bar. In order to examine whether the size of the data set is sufficient, or whether additional training data would improve the answering performance, we provide a learning curve in Figure 1 (right). We use a confidence threshold of $\theta = 0$; since every question is answered, precision equals recall. We display the precision/recall of the answering performance for the supervised classification method described in Section 4.2. We can see that the curve approaches saturation

for the available 805 training instances. This curve, as well as all succeeding diagrams, are based on 10-fold cross validation.

Figure 1 (right) shows a maximum recall of 42%. In order to judge whether this is a “good” or a “poor” result, let us take a closer look at our data set. We study which of all $\frac{19 \cdot 18}{2} = 171$ pairwise combinations of $S_1, S_2 \dots, S_{n-1}$ are separable. We learn a binary SVM decision function for each pair of classes (using 10-fold cross validation) and measure the separability of each pair in terms of its *AUC performance*. The area under the *receiver operating characteristic (ROC) curve* of a binary decision function is just the probability that, when we draw one positive and one negative example at random, the decision function assigns a higher value to the positive than to the negative example.

Figure 2 shows a graph in which each class is a node, and each inseparable pair (indicated by a low AUC value) is an edge. The different line styles represent different AUC value ranges. We notice that there are several AUC values below 60%. These critical pairs are cases in which distinct answers are sent in response to equivalent questions. For example, the question “*When can I expect delivery?*” can result either in answer “*Order will be shipped tomorrow.*” or “*Order has already been shipped.*” Similarly, the question “*The product you sent doesn’t work. What now?*” can result in the answer “*Please return the product, we will send replacement*” or “*Please discard the product, we will send replacement*”, depending on the value of the item. In these cases, it is impossible to identify the correct answer based on just the text of the question. A possible solution would be to extract a customer identification from the message, and retrieve the additionally needed information by a database query.

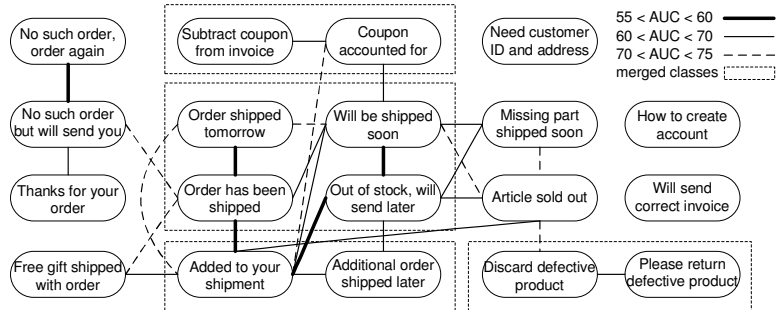


Fig. 2. Inseparability graph: Nodes represent answer classes, edges represent inseparability relations (low AUC values); dashed line boxes show merged classes.

In order to simulate the performance that we would obtain if we had additional information from the order processing database, we merge some answer classes that are based on identical questions but could be distinguished with information from the database. The merged classes are shown in Figure 2 by dashed line boxes. We look at the *precision-recall curves* in Figure 3 (left), where curves are displayed for the merged and original answer classes using the supervised classification method described in Section 4.2. Over the recall levels, this

improves the precision by 10-20%. Figure 2 shows that we still have indistinguishable pairs in distinct sections of the graph. This is the reason why we still do not exceed 50% recall.

We notice that 35% of the inbound messages contain quotes of old messages in addition to the actual question. How do these quotes affect the answering performance? We automatically remove the quotes and compare the performance in Figure 3. We see that removing quotes is slightly detrimental, the quotes seem to contain additional useful information.

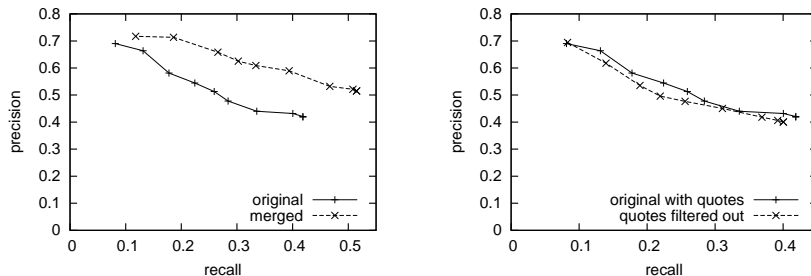


Fig. 3. Precision-recall curves for merged/original answer classes and for question messages with/without quotes.

6.2 Comparison of the different answering methods

We compare the *precision-recall curves* of the two baseline methods of Section 4 to the method for learning from message pairs discussed in Section 5. All presented results for learning from message pairs are based on the averaged precision and recall values over ten different clustering runs to compensate for the randomized cluster initialization. The 10-fold cross validation is nested into ten clustering runs so each value is based on 100 learning cycles.

Figure 4 (left) shows the curves for the data set with *excluded* “miscellaneous” class. Our motivation for excluding the miscellaneous class is that our evaluation metric considers all miscellaneous messages to be answered wrongly which might impose a pessimistic bias. Figure 4 (right) shows the same methods and curves

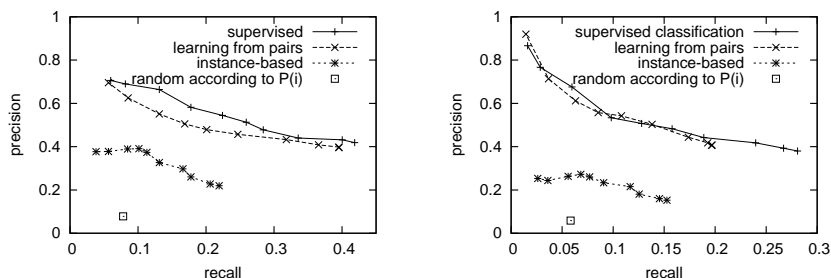


Fig. 4. Precision-recall curves for learning from message pairs and the two baseline methods; message pairs of “miscellaneous” class *excluded* (left)/*included* (right) from training set.

with *included* “miscellaneous” class. In both settings, the learning from message pairs method is by far better than the instance-based method and, needless to say, better than random assignment according to $P(i)$. We notice also that the difference between supervised learning and learning from message pairs is only small. Especially in Figure 4 (right), where “miscellaneous” messages are included, the difference is marginal. This is surprising because for the supervised learning method a user has manually labeled all training messages with one of 19 answer templates. By contrast, our algorithm for learning from message pairs has achieved nearly the same result without any manual labeling.

Additionally, we are interested in the variance of the performance of learning from message pairs as a result of the random influence in the EM-clustering. The standard deviations are rather low, 1.7% (precision) and 1.6% (recall) for $\theta = 0$.

6.3 Influence of clustering parameters

We want to find out to what extent the results depend on optimally chosen clustering parameters σ_τ^2 and π . Figure 5 shows the *precision-recall curves* for different values of σ_τ^2 (right) and π (left). We notice that the performance differences are marginal. Over the range of settings for σ_τ^2 and π , we observe quite diverse clusterings with between 12 and 30 clusters. However, the clustering has only a small influence on the answering performance of the resulting classifier. When a class is split up into two clusters and two distinct classifiers are learned for these clusters, this has only a marginal influence on the resulting performance because confusions between these two do not result in wrong answers.

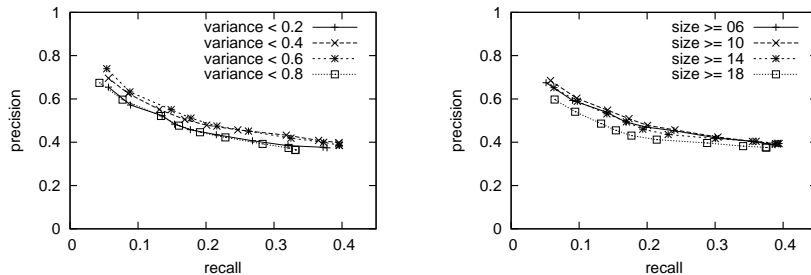


Fig. 5. Precision-recall curves for different variance threshold (left)/pruning (right) parameter values of the clustering algorithm ($\sigma_\tau^2 = 0.4$, left, and $\pi = 10$, right)

7 Conclusion

We identified *message answering* as a new problem setting. The message answering task is to learn a mapping from inbound to outbound messages; the training data consist of message pairs recorded from past communication. This setting is motivated by a practically relevant application that is not appropriately subsumed under the classification framework: email communication in service centers. Rather than a concluding solution, this paper provides a starting point for discussion of the message answering problem.

We first used a supervised classification approach which requires a user to label all message pairs with a small set of answer templates. An instance based approach that retrieves the answer to the most similar question served as lower baseline. Our solution clusters the answer messages, and then uses the inbound messages associated to each cluster of outbound messages as positive examples for a support vector classifier. Our case study showed that the classification based approach achieves only marginally higher precision and recall values (but a higher maximum recall) than our solution for learning from message pairs. The instance-based method, by comparison, performs substantially worse.

Precision and recall were only marginally influenced by the choice of clustering parameters, even though the clusterings obtained for distinct parameters were diverse. A small but significant additional result is that removing quotes from inbound messages deteriorates the answering performance.

We see two options for integrating the presented method into an actual answering system. One is to automatically answer high confidence messages and route low confidence messages to a human agent. A second option is to integrate the predictor into an email client which would then propose answers for incoming emails; these answers can be manually accepted, rejected or modified.

Whether the precision/recall values that we observed are sufficient for practical applications depends on several factors – such as whether the goal is to interactively generate a proposal or whether the emails are actually answered automatically, as well as the volume of emails received. In order to quantify the actual user benefit and economic impact, we will conduct an additional field study.

In the service center setting of our data set, answers do not need to incorporate specific details of the question; *e.g.*, the question “What should I do with the defective *screwdriver set?*” results in the general answer “Please return defective *product.*” There might be settings in which an answer needs to repeat specific information of a question. This case is not covered by our current solution; here, an analysis of repeated phrases in question and answers during training and an additional information extraction step for answering a new question are needed.

The data set that we studied contains no questions with multiple topics; but for other settings, this might be different. An additional challenge is to interpret the question and answer messages as a mixture of multiple question and answer topics and allow the answering hypothesis to identify different question topics and generate a mixed answer.

One might argue that, if answer topics are stable over time and their number is small, a manual labeling of training messages is manageable. But first of all the answer topics might not be stable in practice and secondly, even if a manual labeling of several hundreds of emails sounds easy to us, it would prevent many users from using such a system. A “plug & play” approach like our learning from message pairs seems much more promising for fast acceptance in practical settings.

Acknowledgment

This work has been supported by the German Science Foundation DFG under grant SCHE540/10-1.

References

1. I. Androutsopoulos, J. Koutsias, K. Chandrinou, and C. Spyropoulos. An experimental comparison of naive Bayesian and keyword based anti-spam filtering with personal email messages. In *Proceedings of the International ACM SIGIR Conference*, 2000.
2. T. Boone. Concept features in Re:Agent, an intelligent email agent. *Autonomous Agents*, 1998.
3. J. Brutlag and C. Meek. Challenges of the email domain for text classification. In *Proceedings of the International Conference on Machine Learning*, 2000.
4. W. Cohen. Learning rules that classify email. In *Proceedings of the IEEE Spring Symposium on Machine Learning for Information Access*, 1996.
5. E. Crawford, J. Kay, and E. McCreath. IEMS - the intelligent email sorter. In *Proceedings of the International Conference on Machine Learning*, 2002.
6. H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5), 1999.
7. N. Ducheneaut and V. Belotti. Email as habitat: an exploration of embedded personal information management. *Interactions*, 8:30–38, 2001.
8. B. Galitsky. *Natural Language Question Answering Systems*. Advanced Knowledge International, 2003.
9. S. Kiritchenko and S. Matwin. Email classification with co-training. Technical report, University of Ottawa, 2002.
10. A. Kolcz and J. Alsepector. SVM-based filtering of e-mail spam with content-specific misclassification costs. In *Proceedings of the ICDM Workshop on Text Mining*, 2001.
11. W. Mackay. Triggers and barriers to customizing software. In *Proceedings of the International Conference on Human Factors in Computing Systems*, 1991.
12. P. Pantel and D. Lin. Spamcop: a spam classification and organization program. In *Proceedings of the AAAI Workshop on Learning for Text Categorization*, 1998.
13. M. Pazzani. Representation of electronic mail filtering profiles: a user study. In *Proceedings of the ACM Conference on Intelligent User Interfaces*, 2000.
14. J. Provost. Naive Bayes vs. rule-learning in classification of email. Technical Report AI-TR-99-284, University of Texas at Austin, 1999.
15. J. Rennie. iFILE: an application of machine learning to email filtering. In *Proceedings of the SIGKDD Text Mining Workshop*, 2000.
16. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk email. In *Proceedings of the AAAI Workshop on Learning for Text Categorization*, 1998.
17. T. Scheffer. Email answering assistance by semi-supervised text classification. *Intelligent Data Analysis*, 8(5), 2004.
18. R. Segal and J. Kephart. Mailcat: An intelligent assistant for organizing mail. In *Autonomous Agents*, 1999.
19. K. Shimazu and K. Furukawa. Knowledge discovery in databases by Progol – design, implementation, and its application to expert system building. In *Proceedings of the ACM Symposium on Applied Computing*, 1997.