Designing an Answer Template Retrieval System

Pedro Mota, Bruno Martins, Luísa Coheur
L^{2}F / INESC-ID Lisboa
Lisboa Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
pedro.mota@l2f.inesc-id.pt

Abstract. This paper describes an Information Retrieval approach to address the problem of automatically retrieving an answer to an input email. This task is to be performed in the costumer service of a company that receives emails from their clients, regarding their products and services. It is common for the employees in the costumer services to use past answer as templates when they receive a reoccurring email about a certain issue. This way, instead of starting to write an answer from scratch, they just need to edit some parts of an old answer (such as names or dates), which is much faster. Therefore, what is required is to return an answer which can be used as a template. In this context, two tasks must be performed by our system: identifying similar answers that can be used as templates and retrieving the appropriate answer given a new (unseen) email. Throughout the design of the retrieval system, special attention is given to scalability aspects, since large amounts of data are involved. In this perspective, we present efficient methods that can be used in both tasks that are addressed. Results show interesting values for Precision@10 and Mean Reciprocal Rank@10 (0.62 and 0.45 respectively), considering that no more domain specific processing on the data was done after finding similar answers.

1 Introduction

This work addresses the problem of retrieving the appropriate answer template to a given input email. This task is domain specific and is performed in the context of a costumer service of a company. The costumer service receives a large amount of emails from their costumers, regarding many different subjects related with the company's products and services. What happens when the company's employees answer these emails is that they end up looking for some past answered email, containing a similar answer to the one they need to provide. The explanation for this behavior is that despite the large amount of input emails, many of them regard the same issue and thus all such emails can be given similar answers. In this context, for the employees it is much easier to just copy and paste these similar answers, instead of typing a new answer from scratch. After finding the appropriate answer, it is only necessary to edit some of its' fields. These fields correspond to information that needs to be adapted to the current situation, such as the name of the person to whom the employees are answering or an address, for instance. Therefore, we can see these answers as templates that can be reused. Taking all this into account, the retrieval task we propose to address is to return an answer to a new (unseen) email, which only requires an 10% of editing; this percentage was given by the client company and corresponds to a rough estimate to what is considered similar answers.

A data set of Email/Answer (E/A) pairs was provided by the company and in it the relation of similar answers (by a maximum 10% edit distance) is not explicit. Thus, our first main focus is in designing a way of finding these relations and, in the end, characterize the obtained data in terms of their edit distance relations. Another concern during the development is the large amount of data that the system needs to be able to cope with. Therefore, we need to ensure that the used algorithms will be able to scale the system.

After finding E/A pairs that have a maximum 10% edit distance, we are then able to assess if our retrieval system is able to return the appropriate answer to be used as a template when a new (unseen) email arrives. Therefore, we have the necessary conditions to apply techniques that allow the retrieval task to be performed. In this context, we will also benchmark some of these techniques in order to obtain preliminary results on our data.

The remaining of this document is organized as follows: in Section 2 we present the related work. In Section 3 we describe our approach to finding similar answers problem, and the corresponding evaluation. In Section 4 we detail on the retrieval procedure of our system, and make the corresponding evaluation. Finally, in Section 5, we draw some conclusions and point directions to future work.

2 Related Work

The following sections describe related work which will be taken into account during the development of the retrieval system. Section 2.1 describes various approaches to the Frequently Asked Questions (FAQ) Retrieval problem, which is similar to the answer retrieval problem that we are dealing with. We also describe a technique called *MinHash* (Section 2.2), which will be crucial for addressing the scalability issues in our work.

2.1 Frequently Asked Questions Retrieval

In this section, we describe the previous work done in the task of FAQ Retrieval. This task can be defined as answering users' questions posed in Natural Language by recovering the most relevant Question/Answer (Q/A) pairs from a FAQ compilation [1]. From this definition it is possible to see that this task is similar to the one we propose ourselves to solve. The major difference is that instead of having a Q/A pair we have an E/A pair.

In order to perform FAQ Retrieval there are different tasks which need to be addressed: 1) gathering of FAQ pages, 2) extract the Q/A pairs from the FAQ pages, 3) return the appropriate answer to a new (unseen) user question, and 4) maintaining the FAQ Retrieval system. Taking into account our answer template retrieval task, we only need to address the last two problems (the E/A pairs are already provided

in structured a format). In the task of FAQ Retrieval, and from an analysis of the related literature, we can see that the main challenge lies in overcoming vocabulary mismatches.

In the work of Jijkoun and Rijke [2], the retrieval problem is viewed as a Fielded Search Task. The considered fields are:

- Question text
- Answer text
- Title of FAQ pages
- Full text of FAQ page

The above fields are used in the Vector Space Model (VSM) implementation of Lucene [3]. This type of model uses vector representations of pieces of text, in which the coordinates are, for instance, the frequency counts of the words in the text. Given a user question, the retrieval value of a Q/A pair is calculated as a linear combination of similarity scores between the user question and the fields of the Q/A pair. Several combinations of these fields, as well as other features (such as N-gram versions of the text or the use of a stemmer), were tested. The detailed description of the used models and the corresponding obtained results can be found in [2].

Another approach was taken by Xue et al. [1]. In this approach the goal is to estimate $P(q|(Q/A)_i)$ (the probability of a user question q, given a $(Q/A)_i$ pair). Therefore, we only need to find the Q/A pair that yields the highest probability value. Two previously existing models were used for providing the probability estimation values: Query Likelihood Language [4] and IBM [5]. The first model is based on estimating word probabilities given a certain document and then smoothing the obtained values with the full collection of available documents (using a controllable λ factor). The major drawback of this model is that it gives a zero probability to unseen words. As for the IBM model, it is a translation-based approach in which the probability of translating a word into another one is estimated. The application of this to FAQ Retrieval consists in viewing the Question and Answer parts as two different languages and, thus, we try to translate Questions into Answers. This creates a new problem, which is the fact that the target and source languages are the same and, therefore, the probability of translating a word to itself is very likely to be high (self-translation problem). Another issue with the IBM model is that there is not a controllable smoothing mechanism for the estimated probabilities. Moreover, an advantage of this model is that there is the possibility of translating words into *null*, and this event has an associated probability; this is specially relevant to unknown words, since they can be translated to *null*. Taking into account the pros and cons of both techniques, in the end the final value of $P(q|(Q/A)_i)$ corresponds to a linear combination of both, Query Likelihood Language and IBM models. For a complete description of these models see [1].

The approach developed by Wu et al. [6] is based on two concepts: Question Category Segments (QS) and Keyword Segments (KS). A QS corresponds to a parse-tree that expresses the semantic components of a given question, generated by a grammar in which the rules are semi-automatically generated by an agglomerative clustering algorithm [7]. The obtained parse-tree is then used in a tree similarity algorithm with the questions that are in a FAQ database. When comparing the words in the nodes of the trees, the *HowNet* [8] resource and a thesaurus are used, so that a zero score is not immediately assigned when two words are distinct. In what respects the KS, it represents the most relevant words that are in Questions or Answers. For this end, the unknown word extraction method [9] is used for obtaining the necessary KS. These KS are then used to obtain a score given a user question and a question in the database. This score corresponds to the average of the similarities between the words and their corresponding alignment [6]. The similarity score also takes into account the thesaurus definition of the words. The QS and KS scores are computed for each question in the database, and results are then ranked according to the following algorithm:

1. Rank scores with KS similarity
2. Rerank top-n scores with QS similarity
3. Disambiguate with QS similarity from questions and answers

In the last step, top scores which are equal or very close are disambiguated by computing an additional QS similarity between the user question and the answer of the corresponding Q/A pair. This score is linearly combined with the already computed QS score.

The last work on FAQ Retrieval that we are going to describe was developed by Moreo et al. [10], using Case Based Reasoning. In this approach, instead of using Q/A pairs individually, clusters of these pairs are formed. Each cluster is composed by Q/A pairs in which all questions are reformulations of each other. The individual clusters are called *cases*. This is a major difference from the other approaches, since they only look at Q/A pairs individually. For each *case* that we have, a Minimal Differentiator Expressions (MDE) is extracted, which is basically a pattern that occurs in a particular *case* [11]. The goal is to find MDE which covers most of Q/A pairs in a *case* and exclude the ones in other *cases*. Also, the MDE should be as small as possible. During the process of computing MDE, words can be matched through *WordNet* [12]. According to the characteristics of the obtained MDE, weights are assigned to them. These weights are taken into account when more than one MDE matches the input.

2.2 MinHash

Despite of not being a serious issue in the current stage of development of this work (since our data set is not very large), we know that efficiency problems will occur when dealing with a full data base of E/A pairs. As we will see in Sections 3.1 and 4.1, set intersection operations are crucial. The problem is that this operation is very expensive due to the size of the E/A pairs. In order to cope with this issue, we make use of the *MinHash* technique. The idea of this technique is to compare samples of documents instead of their full versions. If we choose this sample in a naïve way (randomly for instance) we will end up with samples that are not very representative of the documents, hence the resulting comparison will not be reliable. In the MinHash procedure, n hash functions are picked and each of them will hash all the tokens in the documents. The tokens in the documents are hashed to integer values, and, for each function, we choose the minimum hash value obtained. In the end, we will have a set of minimum hash values (one for each function) that represents our sample of the document. The point is that the number of hash functions used is much smaller than the original size of the documents, this way reducing the cost of the set intersection operation. Also note that we can cache the obtained sample and, therefore, compute the minimum hash values only once. For a complete description and proof of MinHash see [13, 14].

3 Finding similar Email/Answer pairs

The following sections describe our cluster based approach to finding similar answers that can be used as templates (Section 3.1), and the corresponding evaluation of the obtained data (Section 3.2).

3.1 Clustering Approach

In this section we describe the method used to determine which answers are similar to each other. We opted for a clustering approach to this task and, thus, the final result of the this stage are clusters of E/A pairs in which each answer of an element of an individual cluster can be transformed into another one (of the same cluster) by editing up to 10% of the answer text. The concrete definition of the used edit percentage function is described in Equation (1). The numerator of the equation basically represents the number of words that need to be substituted, added or removed in order to transform one answer in the other. The $EditPercent$ function is close to a standard $Jaccard$ similarity, the difference is that words that exist only in one of the sets are not so strongly penalized. This has to do with the fact that we can substitute some word that exists only in one set, by another word that also only exists in the other set. The advantage in our case is that for the fields in the answer contribute less for the dissimilarity.

$$EditPercent(A_i, A_j) = \frac{Max(|A_i| - |A_i \cap A_j|, |A_j| - |A_i \cap A_j|)}{Max(|A_i|, |A_j|)} \quad (1)$$

With Equation (1) we have a definitive way of knowing which answers in the data set are similar, which gives a strong restriction on the answers can be used as templates. Therefore, we can use a simple clustering algorithm in order to generate clusters with similar answers. The pseudo-code of the implemented algorithm is described in Algorithm 1. In the algorithm, the $clustersForMerge$ variable represents all clusters to which the current pair ($pair_i$) is at a maximum of 10% edit percentage. The idea is to keep track of these clusters (line 9). If the current pair cannot be added to any existent cluster we create a new one (line 21). Otherwise, we added it to one of the possible cluster (we always add to the first cluster in the list, line 16). Finally, we also check if the pair had multiple cluster possibilities and merge those clusters into a single one (line 18). From Algorithm 1 we can see that the bottlenecks mainly come from two aspects: we might have to always compare each pair with the remaining and the $EditPercentage$ function requires expensive set intersection operations. Currently we only deal with the later issue, through the use of the MinHash technique.

Although we have a measure that allows us to assess if two answers are equal, the vocabulary mismatch problem is still an issue. This means we might have similar answers but that are not at a 10% edit distance. The causes for this phenomena are: Named Entities (NEs) and each employees' own way of writing an answer to an email. In what respects the vocabulary mismatch problem caused by NEs, the following ones were identified in the corpus: proper nouns, locations, dates, numerical expressions (quantities for instance) and addresses. The current approach to this problem is based on regular expressions which can capture these NEs. It should be noticed that this approach has limitations, in the sense that not all NEs are captured (for instance, only the proper names that are in beginning of an answer are caught). The second issue is related with the fact that there are not strict answers that the employees need to give in a particular situation. Despite this, there are still a significant parts of answers (given by different employees) that are similar. The differences are in some expressions such as greetings, or the way of starting or ending an answer. To deal with this problem we eliminate these expression by looking at the answer structure (we can easily identify the greeting and ending this way) and again using regular expressions (for the answer starting part). Both NEs and these expressions are removed from the document prior to making the comparisons.

Algorithm 1 Edit Percentage Clustering

```
1:  for (pair_i : E/A-pairs) do
2:      if (clusters.size == 0) then
3:          clusters.add(new Cluster(pair_i, ID));
4:          ID++;
5:      else
6:          for (cluster : clusters) do
7:              for (pair_j : cluster) do
8:                  if (EditPercent(pair_i, pair_j) <= 0.1) then
9:                      clustersForMerge.add(cluster.ID);
10:                     break;
11:                 end if
12:             end for
13:         end for
14:     end if
15:     if (clustersForMerge.size >= 1) then
16:         clusters.get(clustersForMerge.get(0)).add(pair_i);
17:         if (clustersForMerge.size > 1) then
18:             mergeClusters(clustersForMerge);
19:         end if
20:     else
21:         clusters.add(new Cluster(pair_1, ID));
22:         ID++;
23:     end if
24: end for
25: return clusters;
```

3.2 Clustering Evaluation

As stated before, one of the main concerns during the development of our Retrieval System is its scalability. In this context, we want to make use of MinHash versions of the answers in the clustering procedure. These versions can be obtained by varying two parameters in the MinHash technique: the type of the used hash function and the number of functions. Therefore, we need to determine these parameters, in a way that will guarantee us that the error in the values of set intersections, using MinHash versions of answers, will not be significant. With this in mind, we carried out an experiment on our corpus, in which we computed the Root Mean Square Error (RMSE) (Equation (2)) for the $EditPerct$ function between the values obtained using full versions of A_i ($X_{obs,i}$) and the corresponding MinHash versions A_j ($X_{model,i}$); this was done for all possible E/A pairs.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}} \quad (2)$$

We also measured the used memory during the clustering process and the execution time, to better guide us in the choosing of the parameters for MinHash. The characteristics of our E/A corpus used in this experiment are in Table 1. Taking into account these characteristics, we opted by varying the number of used hash functions by steps of 20 from 10 to 90 functions; this way we can observe the variation of RMSE when using a number of functions that gets closer to the real average length of the answers.

In Table 2 are the results for the clustering process using different combinations of parameters for MinHash; these tests were carried out on a 64 bits machine with a 2.7GHz i7 CPU and 8GB of RAM. From the results we can see that the RMSE values are not ideally low (the error is not negligible) but still they are not very high. As expected, the RMSE generally decreases when a higher number of hash functions is used. But it should also be noticed that the error difference also starts to become very low with the increase of the

	Email	Answer
Quantity	12341	12341
Number of words	660761	901230
Unique words	57289	8946
AVG Number of words	54 $\pm$ 34.3	101 $\pm$ 64.3

Table 1. Characteristics of the E/A corpus.

number of hash functions. For instance, the RMSE difference when using Mumur with 50 and 70 hash functions is just 0.001. In what respects the clustering time and used memory, as it was expected, they increase with the number of used hash functions. Taking into account the obtained measurements in this experiment, we opted for the use of 10 polynomial hash functions. This option is mostly related with the scalability of our system. In this perspective, we believe that, for instance, a decrease of 0.02 in RMSE (using 30 polynomial hash functions) does not compensates a 76% increase in the time and a 192% increase in used memory. Also, we think that such low differences in RMSE are not enough to bias the results obtained when using set intersections operations. Comparing the performance of the clustering algorithm using the MinHash and the full answers, we can see that huge gains are obtained; the amount of used memory with MinHash answers is 82.5 times less, and computations are 1396.8 times faster.

Hash Function	Number of Functions	RMSE	Time (s)	Memory (MB)
Polynomial	**10**	**0.14**	**2.5**	**2.4**
	30	0.12	4.4	7
	50	0.117	6.4	11.8
	70	0.119	8.6	16.5
	90	0.117	10.6	21.2
Murmur	10	0.15	2.5	2.4
	30	0.121	5.1	7
	50	0.117	7.7	11.8
	70	0.116	10.3	16.5
	90	0.113	13	21.2
Murmur3	10	0.158	2.2	2.4
	30	0.123	4	7
	50	0.118	6	11.8
	70	0.117	7.9	16.5
	90	0.116	9.9	21.2
Linear	10	0.201	2.2	2.4
	30	0.139	4.6	7
	50	0.132	5.9	11.8
	70	0.122	7.7	16.5
	90	0.123	9.6	21.2
None (Full answers)	-	0.000	58.2 (min)	198

Table 2. Results for the different parameter combinations in MinHash.

After having defined the full procedure for obtaining the clusters from our data, we will now characterize them. In Figure 1 we can observe the number of emails that each individual cluster has. From the curve of the graph we can conclude that there are few clusters with a high number of emails (the maximum is 610) and that there is a long tail for the low number of emails, indicating that most clusters have few emails in them (the average number of emails in the clusters is 2 ± 10.9 and the median is 1). Also, the number of obtained clusters is 6145.

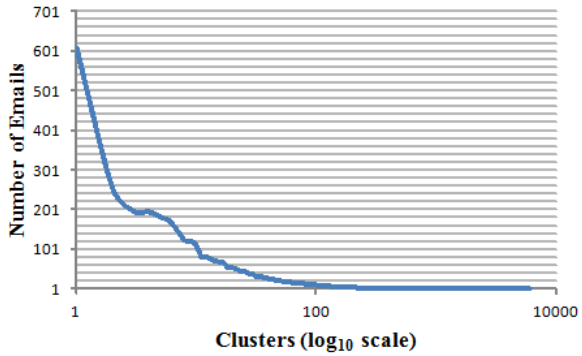

Figure 1. Number of emails in each individual cluster.

In order to have a more concrete notion of the distribution of E/A pairs in the clusters, Figure 2 presents a plot with the number of clusters with a particular amount of E/A pairs. From the plot we can see that this distribution is completely dominated by clusters with single E/A pairs (4500 out of 6145). Also, as the number of E/A pairs increases, the number of clusters decreases drastically (this decrease is already abrupt for 3 E/A pairs). The average number of clusters is 112 ± 678.9 and the median is 2.

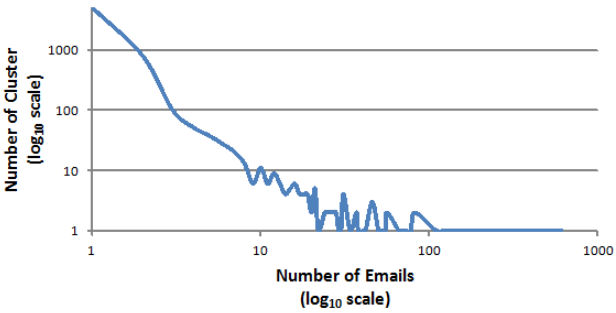

Figure 2. Number of cluster with a certain quantity of E/A pairs.

One possible explanation for this data distribution is that we might have E/A pairs which are close to the 10% boundary, and these could actually be suitable for merging with some other cluster; recall that the 10% edit percentage is just a rough estimate. In order to see if this happens with our data, a contingency matrix was built. The rows and columns of the matrix correspond the the obtained clusters and the values are the lowest edit percentage between two E/A pairs in those clusters (single-link clustering). With this matrix we obtained the plot of Figure 3. In this type of plot, the darker the shades are, the more close are the clusters; this is the case of the diagonal of the matrix, which corresponds to the edit percentage of a cluster and itself. Since the majority of the shades are white or light gray (excluding the diagonal), we can conclude that the clusters are indeed distant from each other. Therefore, we conclude that actually have some type of answers that appear only once in our corpus. Also, a similar plot was obtained when using the highest edit percentage values in the cells of the matrix. The main difference is that the diagonal is not completely black, but still, the grand majority has dark shades of gray, which

indicates that the pair within the clusters are close to each other.

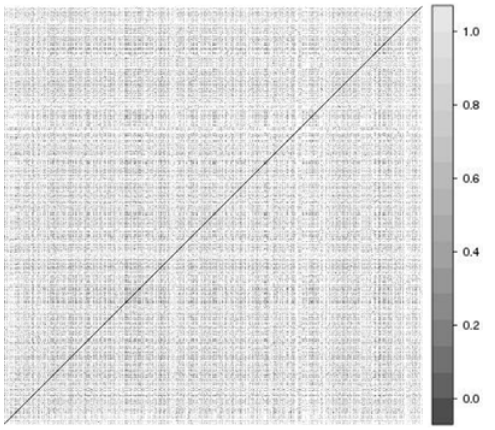

Figure 3. Level plot of the contingency matrix of the E/A corpus.

4 Answer Template Retrieval

Our approach to the answer template retrieval task consists in viewing it as classification task. In Section 4.1 we describe such approach and the used techniques, and in Section 4.2 the corresponding evaluation is carried out.

4.1 Modeling Retrieval as a Classification Process

This section describes the approach taken to the task of receiving a new (unseen) email and retrieving the corresponding answer template to be used.

After the clustering procedure, we obtain groups of emails which are at a 10% edit percentage of each other (for their corresponding answers). This allows us to define the retrieval task as a classification problem. This happens because if we use an email from a cluster as a test instance, and we then can classify it correctly (determine its cluster group), we can retrieve any of the answers in the E/A pairs of that cluster. Therefore, we can make use of any algorithm which corresponds to a classifier. Taking this into account, we used the Language Understanding Platform (LUP) [15], which is able to receive our cluster corpus and test different classification algorithms on it. The available algorithms are:

- Support Vector Machines (SVM) [16]
- Logistic Regression [17]
- Cross Entropy [18]
- K-Nearest Neighbors
- Important Words Match (IWM)

The K-Nearest Neighbors algorithm consists in using a similarity measure to obtain a value that tells us how similar two documents are (two emails in our case). This value is calculated for each email in our corpus, and a Top-K list of the emails with the highest similarities is obtained. From this list we count the most predominant cluster (the one with the higher number of emails present in the list) and return it. If a tie exists, we return the cluster which has the highest similarity value. This procedure is close to the one described in Algorithm 1, and the major difference is that instead of using Equation (1), we use a similarity measure. In LUP the following string similarity measures are available: Dice, Jaccard, Overlap and VSM. It is also possible to linearly combine the Jaccard and Overlap measures and use $tf\text{-}idf$ counts. The retrieval model used in the VSM uses the cosine similarity measure and a linear combination between the similarity of the email and answer parts. It should also be noticed that some of the considered string similarity measures require set intersection operations. Therefore, we can again use MinHash representations of the emails.

In what respects the IWM technique, a list of important words is calculated per cluster; the size of the list is configurable. These words are chosen according to $tf\text{-}idf$ values calculated from the emails in a cluster and in the full E/A corpus. The final list is sorted according to the corresponding $tf\text{-}idf$ values. When a new (unseen) email comes, for each cluster, we iterate the corresponding important word list and see from it which words are in the input email. The score for a cluster is the sum of the index position in which we found an important, and the cluster with the highest score will provide the classification.

4.2 Classification Evaluation

The carried out evaluation followed a 5-fold cross validation procedure (that is, the corpora was divided in 5 random partitions and classification techniques were trained with 4 partitions and tested with the remaining one). At the end, an average of the results was made. By using this methodology we estimate how well we would do in a real use situation of the retrieval system.

In Section 3.2, a large number of clusters with a single E/A pair was reported. These clusters were not considered for this evaluation, since if they end up in a test partition there will not be a train pair to match, and if they are in a train partition, no test instance will be present in that fold. After removing these clusters, the corpus of E/A pairs has the characteristics described in Table 3.

	Email	**Answer**
Number of clusters	1150	1150
Quantity	7346	7346
Number of words	279203	701230
Unique words	25162	6946
AVG Number of words	38 ± 20.5	95 ± 55.2

Table 3. Characteristics of the E/A corpus used in the cross-validation.

Before carrying out the cross-validation evaluation, the emails parts of the pairs of the corpus were normalized by transforming all the text in low caps and removing punctuation and stop words. The results for the cross-validation procedure for the available techniques in LUP are in Table 4. From them we can see that the best performing technique was SVM. The accuracy value for the K-Nearest Neighbors (K = 1) algorithm was obtained by using a similarity measure that combines Jaccard and Overlap (0.8 and 0.2 weights respectively) and $tf\text{-}idf$ values as word frequency values.

As mentioned before, the K-Nearest Neighbors technique can be used efficiently used, due to the use of MinHash samples, for the retrieval task we need to perform; this performance can be further increased if we address the number of E/A pairs comparisons (Section 3.1). It should also be noticed that this technique does not require a model training operation (like SVM or Logistic Regression). Therefore, the only overhead which needs to be accounted for is the assignment of new E/A pair to one of the existing cluster (no retraining of

Classification technique	Precision@1
SVM	0.44 ± 0.012
K-Nearest Neighbors	0.38 ± 0.003
Logistic Regression	0.36 ± 0.007
IWM	0.31 ± 0.007
Cross Entropy	0.11 ± 0.003

Table 4. Cross-validation results for the considered classification techniques.

some kind is required) and, as we saw in Section 4.1, this can be done efficiently. This issue is critical, even when using the current corpus that is not very large, since the execution time (training and testing of the 5 folds) for K-Nearest Neighbors took 21.3 minutes (only testing), whereas for SVM the time was 51.1 minutes for training and 99.7 minutes for testing; this shows the huge overhead of the model training, and also, even for the testing part, K-Nearest Neighbors performs better. Thus, for scalability issues, we focus on trying to boost the K-Nearest Neighbors technique. To guide ourselves in finding the most relevant problems in terms of classification errors, we plotted a graph showing the number of wrong classifications for each individual cluster (Figure 4) for the K-Nearest Neighbors technique. From this graph we can see that there is a low number of clusters in which a great deal of misclassifications happen. Therefore, these particular cases are the ones that deserve a first closer analysis. We manually investigated clusters with 10 or more misclassifications, which represents 0.06% of the total clusters and 16% of the misclassifications.

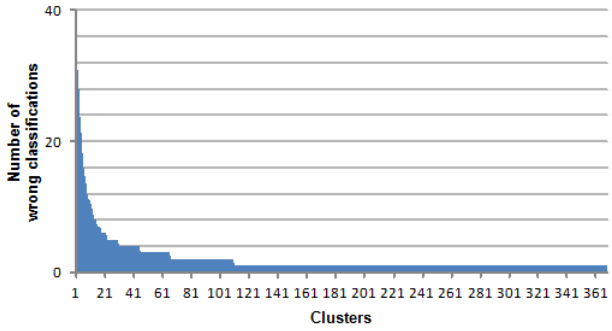

Figure 4. Distribution of the number of wrong classification in the clusters for the 1-Nearest Neighbor technique.

From the analysis of the aforementioned clusters, several issues were identified. The most critical of them is the fact that in some clusters the emails are not semantically related (they refer to different subjects); in [19] a similar problem has been reported. This happens because, in many situations, the costumer service is not able to solve a certain problem and needs to forward it to more specialized employees or tell the costumer to go in person to the company. Therefore, we end up with a set of different generic template answers in which the corresponding emails are very dissimilar. A particular case of this generic template problem, is one in which the answer just states that the problem is solved and the costumer can contact the company again if he has further questions. The problem here is that the corresponding email still has the original question of costumer. The last identified problem that caused some wrong classifications was the fact that some cluster needed to be merged, since their emails do relate with a common subject. The explanation for this is that the answers were given by different employees. This makes the classification task harder, since the emails in different clusters are very likely to be similar.

We also analyzed the obtained accuracy results when varying the K number of neighbors and computed two other measures: Precision@10 (only requires to have the right cluster in a list of 10 possibilities) and Mean Reciprocal Rank (MRR) (Equation (3) also using a candidate list of size 10). By doing this we are assessing how well our retrieval system would do in the task of returning a list of ordered answer template candidates, from which the employee could choose from.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{3}$$

The obtained results are in Table 5 and we can conclude that the Precision@1 results generally decrease with the increase of K. What happened was that the correct cluster did not have many E/A pairs in the K-Nearest Neighbors list because it has few members, thus, the other (wrong) clusters with more members were having more candidates in the Top-K list. This was specially problematic with the clusters with the generic type of answers, since their emails refer some products in services present in other (correct) clusters and they have many members.

Classification technique	Precision@1	Precision@10	MRR@10
1-Nearest Neighbors	0.38 ± 0.007	0.62 ± 0.014	0.45 ± 0.008
10-Nearest Neighbors	0.21 ± 0.006	0.63 ± 0.010	0.33 ± 0.012
40-Nearest Neighbors	0.22 ± 0.006	0.47 ± 0.009	0.30 ± 0.006
100-Nearest Neighbors	0.20 ± 0.004	0.48 ± 0.010	0.29 ± 0.004
500-Nearest Neighbors	0.16 ± 0.005	0.42 ± 0.008	0.24 ± 0.005

Table 5. Cross-validation results for the K-Nearest Neighbors technique.

The Precision@10 results show that we were able to place the correct answer to be used as a template in the top 10 results in 63% of the test cases. The best MRR@10 result was 0.45, indicating that, in average, the correct answer is in the second position. It should be noticed the K with the highest Precision@10 score was not the one with the best MRR@10. The difference in the later measure is more significant, thus, using 1-Nearest Neighbor would be a better option.

5 Conclusions and Future work

In this paper we described the methodology for designing a scalable retrieval system for addressing the problem of retrieving answers to new emails, namely answers that can be used as templates.

One of the tasks that was required to be performed was the identification of similar answers. The main issue in this task is the vocabulary mismatch problem, which was caused by NEs and the employees' own way of writing answer. In this context, a set of regular expression was built and allowed the normalization of corpus. Also, due to the large amount data that has to be dealt with, we used MinHash versions of the answers. In order to determine the best parameters (hash function to use and the number of functions) we calculated

RMSE, execution time of the algorithm for finding clusters of similar answers and the corresponding memory usage. The results showed that using 10 polynomial hash functions gave the best balance between these three values. After this step, we analyzed the obtained clusters. This analysis showed that our data is dominated by clusters with few members. The corresponding contingency matrix showed that the clusters were distant from each other, indicating that we have many answers that were only given once.

The retrieval part of our system was implemented as a classification task, and some first experiments were also carried out. Taking into account that no more special processing of the emails parts was done, the obtained results were very encouraging. For the 1-Nearest Neighbor technique we obtained 0.38 in Precision@1, 0.62 in Precision@10 and 0.45 in MRR@10. The analysis of the results showed that one of the main problems is the fact that we have clusters with many unrelated emails (generic answers) and this made the classification task significantly harder. Another issue is related with the characteristics of the current corpus, which has an unbalanced distribution of E/A pairs in the clusters, which is possibly impairing the K-Nearest Neighbors algorithm.

The developed work allowed us to have some insights about the task at hands, and revealed several issues to be solved in future work. One of the improvements that should be done in our system is using a method other than manually building regular expressions for capturing NEs (a statistical NEs Recognizer could be used for instance); this would allow us to capture NEs in a much more generic way and further normalize our data, which is crucial for the clustering process. Also in this data normalizing perspective, we are using again regular expressions for capturing some parts of the answers that were written in a particular way by a certain employee. It be would be interesting to perform the clustering on a corpora in which we had E/A pairs of the same employee; this is important because in a deploy situation of our system we will have much more E/A pairs from other employees and we would need to manually identify those expressions for them also. Another priority is to deal the clusters that have many unrelated emails. A possible approach is to use two levels of classification, in which we have two corpora: one with just the generic type E/A pairs and the other with the remaining. Then three classifiers need to be used. One for identifying if an E/A pair is of the generic type or not and the other would be a specialization of each of these cases; a similar idea is described in [20] and obtained a substantial improvement. Since we have implemented efficient procedure throughout our retrieval system, we are now interested in make another set of experiments with a much larger corpus of E/A pairs; it is likely that this way we obtain a much more balanced distribution of E/A pairs in the clusters and, thus, obtain better classification results. Another possibility for improving the obtained results, is to use weighted n-gram versions of the MinHash representations of the E/A pairs, as described in [21]. Finally, we also want to address the bottleneck of having to make many comparisons between E/A pairs (both in the clustering and in the classification); for this purpose we can use the two stage Locality Sensitive Hashing approach, as described in [22].

References

[1] Xue, X., Jeon, J., Croft, W.B.: Retrieval models for question and answer archives. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '08, New York, NY, USA, ACM (2008) 475–482

[2] Jijkoun, V., de Rijke, M.: Retrieving answers from frequently asked questions pages on the web. In: Proceedings of the 14th ACM international conference on Information and knowledge management. CIKM '05, New York, NY, USA, ACM (2005) 76–83

[3] : Appache lucene: A high-performance, full-featured text search engine library. http://lucene.apache.org. In: Proceedings of the 14th ACM international conference on Information and knowledge management

[4] Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)

[5] Berger, A., Lafferty, J.: Information retrieval as statistical translation. In: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '99, New York, NY, USA, ACM (1999) 222–229

[6] Wu, C.H., Yeh, J.F., Lai, Y.S.: Semantic segment extraction and matching for internet faq retrieval. IEEE Trans. on Knowl. and Data Eng. **18**(7) (July 2006) 930–940

[7] Meng, H.H., Siu, K.C.: Semiautomatic acquisition of semantic structures for understanding domain-specific natural language queries. IEEE Trans. on Knowl. and Data Eng. **14**(1) (January 2002) 172–181

[8] Zhou, Q., Feng, S.: Build a relation network representation for hownet. In: Proc. Int'l Conf. Multilingual Information Processing. (2000) 139–145

[9] Lai, Y.S., Wu, C.H.: Unknown word and phrase extraction using a phrase-like-unit-based likelihood ratio. Int. J. Comput. Proc. Oriental Lang. **13**(1) (2000) 83–95

[10] Moreo, A., Romero, M., Castro, J.L., Zurita, J.M.: Faqtory: A framework to provide high-quality faq retrieval systems. Expert Syst. Appl. **39**(14) (2012) 11525–11534

[11] Moreo, A., Navarro, M., Castro, J.L., Zurita, J.M.: A high-performance faq retrieval method using minimal differentiator expressions. Know.-Based Syst. **36** (December 2012) 9–20

[12] Miller, G.A.: Wordnet: A lexical database for english. Communications of the ACM **38** (1995) 39–41

[13] Broder, A.Z.: On the resemblance and containment of documents. In: In Compression and Complexity of Sequences (SEQUENCES 97, IEEE Computer Society (1997) 21–29

[14] Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. Journal of Computer and System Sciences **60** (1998) 327–336

[15] Mota, P., Coheur, L.: Natural language understanding as a classification process: report of initial experiments and results. (2012)

[16] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)

[17] Carpenter, B.: Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. technical report, alias-i. available at http://lingpipe-blog.com/ lingpipe-white-papers (2008)

[18] Rubinstein, R.Y., Kroese, D.P.: The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-carlo Simulation (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2004)

[19] Mota, P., Coheur, L., Curto, S., Fialho, P.: Natural language understanding: from laboratory predictions to real interactions. In: Proceedings of the 15^{th} International Conference on Text, Speech and Dialogue (to be published), Springer-Verlag (2012)

[20] Patel, R., Leuski, A., Traum, D.: Dealing with out of domain questions in virtual characters. In: IVA 2006. LNCS (LNAI, Springer (2006)

[21] Chum, O., Philbin, J., Zisserman, A.: Near duplicate image detection: min-hash and tf-idf weighting. In: Proceedings of the British Machine Vision Conference. (2008)

[22] Hoffart, J., Seufert, S., Nguyen, D.B., Theobald, M., Weikum, G.: Kore: keyphrase overlap relatedness for entity disambiguation. In: Proceedings of the 21st ACM international conference on Information and knowledge management. CIKM '12, New York, NY, USA, ACM (2012) 545–554