I.J. Information Engineering and Electronic Business, 2012, 1, 1-9
Published Online February 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2012.01.01

Sentence Clustering Using Parts-of-Speech

Richard Khoury
Department of Software Engineering, Lakehead University, Thunder Bay (ON), Canada
Email: Richard.Khoury@lakeheadu.ca

***Abstract*—Clustering algorithms are used in many Natural Language Processing (NLP) tasks. They have proven to be popular and effective tools to use to discover groups of similar linguistic items. In this exploratory paper, we propose a new clustering algorithm to automatically cluster together similar sentences based on the sentences' part-of-speech syntax. The algorithm generates and merges together the clusters using a syntactic similarity metric based on a hierarchical organization of the parts-of-speech. We demonstrate the features of this algorithm by implementing it in a question type classification system, in order to determine the positive or negative impact of different changes to the algorithm.**

***Index Terms*—natural language processing, Part-of-speech, clustering**

1. Introduction

Clustering algorithms are used in many Natural Language Processing (NLP) tasks, from grouping words [1] and documents [2] to entire languages [3]. They are popular tools to use to group similar items together and discover the links between them. In this exploratory paper, we propose a new clustering algorithm to automatically discover and group together similar sentences based on the sentences' part-of-speech syntax. As a clustering distance metric, we use a part-of-speech hierarchy which we developed in our past work [4]. We apply this clustering system to the task of discovering the informer spans that Krishnan et al. proposed in their work on question type classification [5], and our experimental results demonstrate that our clustering system generates interesting results relative to that application.

The remainder of this paper is organized as follows. In Section 2, we review a representative sample of work done on NLP clustering system and in the task of question type classification, in order to clearly illustrate the nature of our contribution. The theoretical frameworks of our clustering algorithm and of our part-of-speech hierarchy are all presented in Section 3. Our ideas have all been implemented and tested, and experimental results are presented and discussed in Section 4. Finally, we offer some concluding remarks in Section 5.

2. Background

Clustering systems are already being used in several branches of Natural Language Processing (NLP). For example, the authors of [3] designed a spoken language recognition system that clusters spoken utterances based on their spectral features. Preliminary tests of their system showed comparable results to the baseline, which is encouraging given the large range of currently-unexplored refinements they could implement. In parallel, the authors of [6] proposed a language recognition system that works by clustering languages and then dividing them hierarchically into sub-clusters. These authors studied several distance metrics to measure the difference between languages, and finally settled on a fusion of measures. In this context, one could see the methodology presented in this paper as the development of a new distance metric based on parts-of-speech, to be used independently or with others in clustering algorithms. Another popular NLP task sometimes done with clustering is that of document topic identification. This one can be done more directly, by representing documents as word vectors and clustering the vectors according to their Euclidean Distance, Cosine Similarity, Jaccard Coefficient, or any other of the many possible vector distance metrics available [2], [7], or with probabilistic measures such as the Gaussian mixture model [8]. This further demonstrates how natural language text can be converted to a form usable by classical mathematical clustering algorithms. A final example, a little more closely related to this research, is the project on part-of-speech clustering presented in [1]. The author's intuition is that they could improve on a unigram language model by differentiating between topic words, style words, and general words based on their parts-of-speech. To accomplish this, they developed a probabilistic clustering algorithm to group a set of 90 parts-of-speech into these categories using two corpora of text documents covering several different topics. Their system discovered appropriate clusters, which could then be successfully applied in a large vocabulary speech recognition system. While our project's aim and clustering technique are different, their work does confirm that parts-of-speech clustering can be applied successfully in NLP applications.

The clustering system we are developing in this research will be customized to learn clusters useful for the task of question type classification. Question type (or answer type) classification is the NLP task of determining the correct type of the answer expected to a given query, such as whether the answer should be a person, a place, a date, and so on. This classification task

is of crucial importance in question-answering (QA) systems, to allow them to retrieve answers of the correct type and to reject possible answers of the wrong type [9]. In fact, it has been shown that questions that have been classified into the correct type are answered correctly twice as often as misclassified questions [9]. Over the years, many varied approaches to question type classification have been proposed. For example, the authors of [10] used a Naïve Bayes classifier to compute the probability of a query belonging to one of six question types given the prior probability of that type multiplied by the conditional probability of the query's features (i.e. the words left after stemming and stopword removal) given that type. Going in a different route, a team from the University of Concordia developed a simple but accurate keyword- matching question type classification system with a large manually-built lexicon as part of their QA system for the TREC-2007 competition [9]. The authors of [11] and [12] tried a similar approach using WordNet as a lexicon, but found that their systems could be misled in cases where the query's syntax affected the meaning and importance of keywords. The authors of [12] took this a step further and added some question syntactic patterns in their system to deal with common problem phrasings. Finally, Zhang and Nunamaker [13] built a pattern-based question type classifier that classifies each user's query according to a set of simple patterns that combine both the wh-terms (who, what, where, when, why, which, whom, whose, how) and the categories of keywords found in the query. It has been pointed out [5] that a common limiting factor in all these systems is the reliance on large unpublished knowledge bases, be it conditional probability tables, keyword lexicons, or lists of syntactic patterns. For our purposes, the most closely-related project is the one proposed by Krishnan *et al*. in [5]. They proposed that the patterns used for question type classification could consist simply of a short string of contiguous words found in the query, which may or may not include wh-terms, and which they called the *informer span* (or simply *informer*) of the query. Their work shows clearly that a support vector machine classifier using hand-made informers yields better results than question bigrams, and that informers discovered automatically work almost as well as hand-made ones. Our work is related to theirs in the sense that, as we will explain later on, the centers of the clusters learned by our system correspond to informers that can be used for question type classification.

3. Methodology

In this paper, we develop a new system for sentence clustering based on the idea of informers and on a part-of-speech hierarchy that we present below. The objective of this algorithm, as for any clustering algorithm, is to discover the set of clusters that best represent the data. In general terms, this means having high intra-cluster homogeneity, a high inter-cluster heterogeneity, and a small number of clusters. The application we are dealing with in this project is question type classification, because it has proven to be a challenging task to automate, as is evidenced by the number of manually-created solutions presented in Section 2, and because of the direct connection with our past work [14]. In this application, the objectives of the clustering algorithm are to learn the smallest set of clusters to allow the best type classification results.

3.1 Part-of-Speech Hierarchy

A part-of-speech (POS) is commonly defined as a linguistic category of lexical items that share some common syntactic or morphological characteristics. However, it is telling that, despite the concept of a POS being thousands of years old, grammarians and linguists still cannot agree on what exactly the shared characteristics are. This question has very real practical consequences: depending on where the line is drawn between common characteristics and distinguishing differences, one can end up with anywhere from the eight parts-of-speech defined in English textbooks to the 198 parts-of-speech of the London-Lund Corpus of Spoken English [15].

Our solution to this problem is to organize lexical items not into a single set of parts-of-speech but into a part-of-speech hierarchy. The lower levels of this hierarchy feature a greater number of finely-differentiated parts-of-speech, starting with the Penn Treebank POS tags at the lowest level, while the higher levels contain fewer and more general parts-of-speech, and the topmost level is a single all-language-encompassing "universe" part-of-speech. Another innovation has been the inclusion in our hierarchy of several "blank" parts-of-speech, to represent and distinguish between the absences of different types of words. In total, our hierarchy contains 165 parts of speech organized in six levels. We originally developed it in the context of a keyword-extraction project; a detailed description of the hierarchy can be found in the paper describing that project [4].

We can define the semantic importance of different lexical items by assigning weights on the connections between POS in the hierarchy. This allows us to specialize the hierarchy for use in different NLP tasks. In our earlier work on keyword extraction [4], weight was given to verbs and nouns – the typical parts-of-speech of the keywords we were looking for. For question type classification, however, verbs and nouns are not the most semantically important words to take into account. Indeed, Tomuro [11] has shown that question type classification relies mostly on closed-class non-content words. The semantic weight in our hierarchy was thus shifted to the subtrees corresponding to popular query adverbs and pronouns, including wh-terms, and calibrated using queries from the 2007 TREC QA track [16] as examples. The value of each POS in our hierarchy is then computed on the basis of its semantic weight and of the number of descendents it has. The resulting hierarchy, with the value of each POS, is presented in Figure 1.

We showed in [4] how using a part-of-speech hierarchy makes it possible to mathematically define several linguistic comparison operations. It is possible to compute the similarity between two words or POS simply as a function of their distance in the hierarchy. This computation takes several factors into account, including the number of levels in the hierarchy that need to be traversed on the path from one POS to the other and the value of each intermediate POS visited. Likewise, we can find a general place-holder POS to represent two words simply by finding the lowest common ancestor of both words in the hierarchy, and the similarity of the place-holder compared to the original two words it represents is again a function of the distance in the hierarchy between each word and the place-holder POS. By extension, we can measure the similarity between two sentences by pairing their words together by similarity; the fact that our hierarchy includes "blank" parts-of-speech means that the two sentences do not need to be of the same length to be compared. And finally, we can merge two sentences by pairing their words together by similarity and replacing each pair by its lowest common ancestor in the hierarchy.

It is now straightforward to see how our part-of-speech hierarchy can be used for the task of question type classification. Given a list of informers representing different question types, we can use the hierarchy to compute the similarity between a user-specified query and each informer. The query is then classified to the same type as the informer it is most similar to.

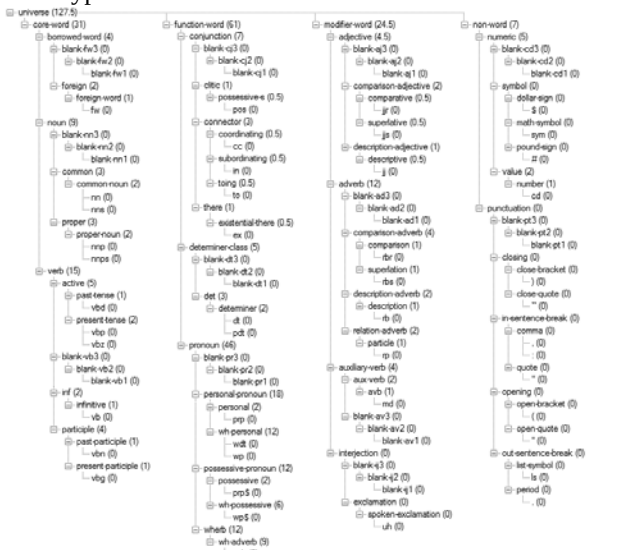

Fig 1: The POS hierarchy (POS values are in brackets).

We implemented such a system in our work presented in [14]. The question types we used are "person" (who), "date" (when), "physical object" (what), "location" (where), "numeric value" (how many/how much), and "description" (how/why). The results presented in that paper show that a classifier based on our hierarchy can achieve an F-measure of 70% using only 16 basic informers, albeit with a strong bias for precision over recall. This stands in stark contrast to the other systems reviewed in Section 2, which need large sets of patterns, lexicons, or probability tables to work well. The 16 basic informers in that project were defined manually, but they are trivial two-word informers that combine each question type's wh-term and one higher-level POS from our hierarchy that represents a past-tense verb (vbd), a present-tense verb (vbz), a description adjective (jj), a singular common noun (nn) or a plural common noun (nns). The informers are listed in Figure 2. Finally, it is worth noting a second result from [14], which is that using a learning algorithm to discover more informers allows the classifier to balance precision and recall without reducing F-measure.

Who [vbz]	What [nn]	How [jj]
Who [vbd]	What [nns]	How [nn]
When [vbz]	What [vbz]	How [vbz]
When [vbd]	What [vbd]	How [vbd]
Where [vbz]	Why [vbz]	
Where [vbd]	Why [vbd]	

Fig 2: 16 basic informer spans from [14].

3.2 Informer-Clustering Algorithm

In our algorithm, each cluster combines together queries of the same type. The cluster's center is defined as the generic query obtained by merging all queries in the cluster. Adding a query to the cluster is done by merging that query with the current cluster center. Splitting a cluster is also possible; when the generic query in the center contains a part-of-speech that is higher than a set threshold in the hierarchy, that part-of-speech is discarded and the query is split in two parts at that point. Each part becomes the center of its own new cluster, combining a copy of the set of queries of the original cluster. Given our discussion in the previous section, it can be seen that the cluster centers are the same as the informer spans we want the algorithm to learn.

The pseudocode of the clustering algorithm we propose is presented in Figure 3. The algorithm takes in as input a list of training queries, which it will try to cluster. Optionally, we may also want to input some manually-defined application-specific informers, such as the 16 basic informers from our work in [14]. This would allow the system to consider clusters that we know to be valid. The impact of this additional input will be studied in the experiments in Section 4.

The algorithm begins by pairing together all queries of each type according to their similarity to each other, with the constraint that each query can only be paired to one other query. The algorithm thus goes through the list of possible pairings, accepting them in order of decreasing similarity, and rejecting those that use queries that are already in more similar pairs. To illustrate, suppose a query type has four queries labeled A, B, C and D. The possible pairings are A-B, A-C, A-D, B-C, B-D and C-D, and the similarities between them (i.e. the distances between them in the POS hierarchy) are 5, 7, 9, 12, 8 and 15 respectively. The algorithm considers these pairs in order of decreasing similarity (increasing distance) starting with the most similar (least distant) pair, A-B. Since neither A nor B has been paired yet, this pair is accepted. The algorithm considers the other pairs in order; however all other pairs except one use either A or B, which is not allowed since they have already been

paired, and thus are rejected. The last pair considered in order is the one with the lowest similarity (greatest distance), C-D. It is however the most similar acceptable pair for these queries, and is retained by the algorithm.

This set of pairs is used by our algorithm as the initial set of clusters. This set will be naturally very large: at least half the size of the initial training set, and more if some of the clusters are split by the method described previously. Moreover, the informers of some of these initial clusters may not be useful to correctly classify queries into question types; they would typically be the result of splitting a cluster into two parts, one of which is irrelevant. For example, imagine two queries that are identical except for a completely different final word before the question mark. Merging these two queries together would give an informer that has a very general part-of-speech instead of the final word, and this informer would in turn be split into two, one that has all the words before the final one, and one that has everything after the final word, which is to say only the question mark at the end. Clearly, a question-mark-only informer is useless for classification! Consequently, the second step of our algorithm is to detect and remove these informers and clusters. This task is implemented by classifying the training set using the list of informers and deleting those informers that lead to more misclassifications than correct classifications, as illustrated in the pseudo-code of Figure 4. This is an optional step, however, since our algorithm can work whether or not it is executed. In our experimental results, we will study the impact of executing or skipping it.

Steps 3 to 6 in our algorithm are the cluster merging steps. In this step, the algorithm considers all pairs of clusters, in order of decreasing similarity of their cluster centers. For each pair, the algorithm merges the cluster centers, and then evaluates the merged center. The quality of the merged cluster is evaluated and, if it is found to be better, the original clusters are discarded and the new cluster is added to the list of clusters for consideration. Otherwise, the new cluster is discarded and the two original ones are maintained. Several metrics could be used for this quality evaluation; in this study we will consider the similarity of the queries to the center and the classification results. In the first metric, we are comparing the quality of the new cluster to that of the original clusters. We compute the average similarity of the queries in each cluster to their respective cluster center to get the quality of the original clusters, and likewise the average similarity of the queries to the merged cluster center gives the new cluster's quality. The cluster with the best average similarity is considered the one with the highest quality. For the second metric, we find the number of training queries that would be classified by the new informer corresponding to the cluster center, and count how many of these queries are correctly classified and how many are not. A cluster whose informer allows more correct classification than incorrect ones is considered to be of good quality. It should be noted that in this second metric, unlike in the first one, a merged cluster is accepted or rejected based on its own performance and not in comparison to the original two clusters it merges together. But either way, the algorithm can use a merger bonus to relax the acceptance criterion and allow the algorithm to accept bad mergers.

The final step of the algorithm consists in a filtering task like the one optionally done in step 2. Earlier in the algorithm it was done to detect cluster that are not useful for type classification and eliminate them from consideration in the rest of the algorithm. Now it is done as a final step, to detect and eliminate similarly non-predictive clusters that have been learned through the repetitive merging and splitting of clusters in steps 3 to 6. This is again done using the filtering algorithm of Figure 4.

```
Input: Training queries, initial
       informers (optional), bonus
1. Pair the training queries by
   similarity to get initial clusters
2. Filter non-predictive initial
   clusters (optional)
3. For each pair of clusters, ordered
   by similarity
    4. Merge the two cluster centers
       together
    5. Evaluate the quality of the
       merger, and apply bonus
    6. If the merger is of good
       quality, save the merged
       cluster and discard the
       original pair; otherwise
       discard the merger and keep the
       original pair
7. Filter non-predictive clusters
```

Fig 3: Pseudocode of our clustering algorithm.

```
Input: Training queries, list of
       informers, Threshold
1. Do:
   2. Classify all training queries
      to the most similar informer in
      the list
   3. For each informer:
      4. NCorrect ← The number of
         queries of the correct type
         classified by this informer
      5. NIncorrect ← The number of
         queries of other types
         misclassified by this
         informer
      6. If NIncorrect*Threshold >
         NCorrect, remove informer
         from the list
7. While informers are removed in
   step 6
```

Fig 4: Pseudocode of the filtering step.

4. Experimental Results

For our experiments, we built a training corpus of queries using the 459 queries from the 2007 TREC QA track [16]. We tagged the words of the queries with their

parts-of-speech using the standard Brill tagger, and we manually classified the queries into their correct question types. We used the data to run five-fold tests, breaking the data into five non-overlapping equal-sized sets of queries and successively training the system with four sets and testing with the fifth.

In each of the experiments, we used the centers of the final clusters generated by the algorithm as informers to classify the queries into each of our six question types, and we computed the precision and recall of the classification using the standard equations given in (1) and (2). We then computed the average precision and recall over all six types, and the average F-measure using equation (3).

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (1)$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (2)$$

$$\text{F} - \text{Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

4.1 Experiments

We thoroughly tested our clustering algorithm by using it to learn informers from our training corpus while varying one or another of the options we described in Section 3.2 and keeping the other options to their "default" settings. This will make it possible to clearly see the contribution and impact of each option we study. For ease of comparison, we listed the default settings of the algorithm in Table 1. The default version of the classifier learns on 69 clusters on average over the five-fold test, and classifying the final fold of training queries using the informers corresponding to the centers of these clusters gives on average a precision of 0.54, a recall of 0.50, and an F-measure of 0.52. However, it is worth noting that the system failed for one fold, by which we mean that the final filtering loop deleted all informers and left an empty set, which naturally gets 0.00 in all classification statistics. If we ignore that fold, the system learned on average 86 informers, and classifies with a precision of 0.68, a recall of 0.63, and an F-measure of 0.65.

Table 1: Default options and settings of the algorithm

Option	Default
Initial informers	None
Filtering initial clusters	Not done
Quality evaluation algorithm	Similarity
Merging quality bonus	1.0
Threshold for final filtering	1.0

4.1.1 Filtering initial clusters

One of the first options mentioned in Section 3.2 is whether or not to filter the initial set of clusters with the algorithm of Figure 4 before running the cluster-merging loop. The intuitive justification for this step is that if the system catches and eliminates non-predictive clusters early on, the merging algorithm will then focus on merging useful pairs of clusters together and learn valid general clusters.

However, the results obtained in this test tell a different story. In our experiments, ignoring the one fold where the system fails and eliminates all clusters, we find that the initial filtering step takes out 28 clusters on average. As expected, after merging, the remaining clusters are more useful, and on average 22 fewer of them are eliminated by the final filtering step. However, this has no measurable impact on the final results of the algorithm. Compared to the default algorithm of the previous section, the final set of informers learned is almost identical with on average only one extra informer learned by the modified algorithm, and the classification results are absolutely identical. This indicates that the clustering algorithm we propose is resilient in the face of initial data of mixed quality, and that the final filtering step alone is enough to eliminate the bad informers learned. An initial filtering of the clusters is clearly unnecessary.

4.1.2 Merged cluster evaluation

In section 3.2 we proposed two methods to evaluate merged clusters and determine whether or not a merger should be kept. The first method uses the part-of-speech hierarchy to compute the average similarity between the clustered queries and the cluster center, and compares it to the similarity in the original clusters to determine if an improvement has been made. The second method finds which queries are classified by the informer at the center of the cluster, and determines whether more correct than incorrect classifications are made. In both cases, we can use a bonus value to allow the merger of bad clusters. The bonus value varies from 0.0 to 1.0. A bonus of 1.0 requires that the merger be strictly an improvement, in the sense that it is either more similar or makes more correct classifications, while a lower bonus value allows some bad clusters, that either have a worse similarity or make more incorrect classifications. We should also note that the algorithm still performs the final filtering step before returning the clusters.

We can compare the two methods in terms of the final number of clusters the algorithm returns and of the f-measure of the classification of the test queries. We begin by considering the number of clusters, presented in Figure 5. It can be seen from this graphic that few clusters are returned at low values of the bonus. In fact, the algorithm is found to fail and return no clusters at all for many of the 5-fold tests in that range. The reason is that, lowering the bonus allows more mergers by tolerating worse and worse mergers. The resulting clusters are bad and taken out by the final filtering step, leading to the many empty sets observed in the results. Good clusters begin to appear and be retained at a bonus

of 0.3 using the classification method and 0.4 using the similarity method, and the number of clusters peaks at 0.6 for the classification method and 0.7 for the similarity method. Finally, the number of clusters spikes for both methods at a bonus of 1.0, where the strict criteria prevents a lot of mergers. Overall, we find that the behaviors of both methods are very similar from this point of view.

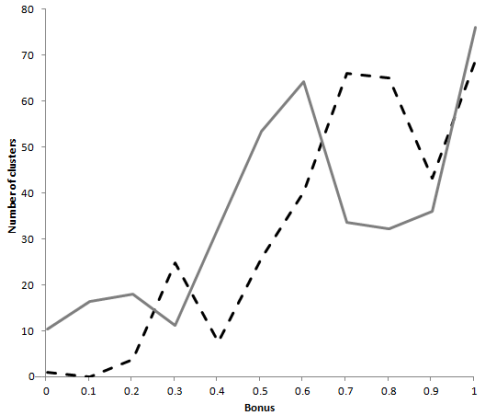

Fig 5: Number of clusters returned using different bonus values with the similarity method (black dashed line) and the correct classification method (solid grey line).

The second comparison point is the average f-measure of the classification using the rules learned for each fold at each bonus value. That one is presented in Figure 6. The left-hand graph of that figure plots the overall f-measure, including the zeros that come from the folds where the algorithm failed. The varying number of zeros at each threshold causes the apparent instability of the results, and overall the shape of the curve follows closely the shape of Figure 5. To truly compare the performance of both methods, it is more informative to consider only those cases where the algorithm didn't fail for either one. The average f-measure of those common cases is presented in the right-hand graph of Figure 6. As we can see in that graph, both methods perform about equally well. The differences visible in the left-hand graph are really only due to different numbers of failed folds.

It is interesting that both methods appear to be equivalent in performance, given that they are very different in implementation: one evaluates the cluster based on classification and the other based on similarity, and one compares the merged cluster to the original ones while the other compares it to its own failure cases. The fact that our overall algorithm performs as well using either methods points to the resilience of the entire system.

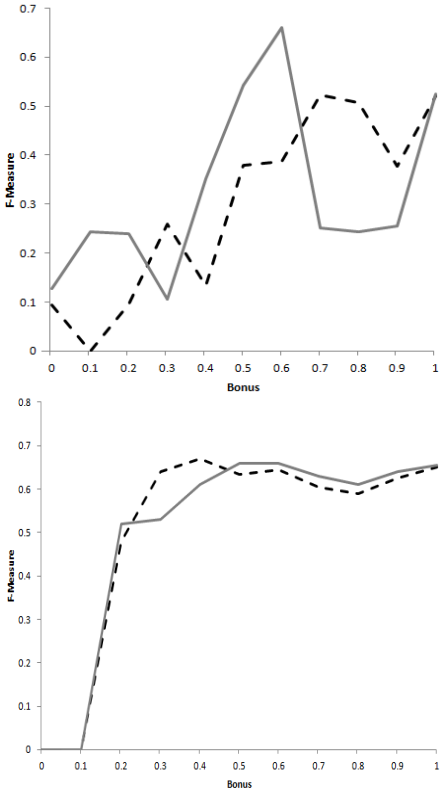

Fig 6: Classification f-measure using different bonus values with the similarity method (black dashed line) and the correct classification method (solid grey line) over all folds (top) and for folds that did not fail on both methods (bottom).

We can note that the second method uses the same evaluation standard as the final filtering step, namely to use the cluster centers' informers to classify the training queries and discard informers that do not lead to enough correct classifications. This leads to a final question: is the final filtering step still necessary, or is it redundant after this training? We ran the test again without the final filtering step to check. The first thing that appears from the results is that this version of the system never fails on a fold; there is no case in which the training ends up eliminating all clusters. This is not completely unexpected, since the filtering step is in charge of eliminating non-predictive informers; without this step, nothing gets eliminated and the algorithm is guaranteed to return something. However, the flip side is that the unfiltered set of informers returned is a lot larger than normal: between 80 and 120 informers, compared to the filtered version that returns on average 34 informers and never goes up to 80. This result is clearly at odds with one of our objectives, which is to generate a set of clusters that is as small as possible. The second thing to consider is whether this larger set of informers allows for better or worse classification of the unseen fifth fold of queries. When compared to the results in the left-hand graph of Figure 6, the classification results with the unfiltered set of informers appear to be a lot better. However, this is due to the fact that the comparison is not entirely fair: as we mentioned, the smaller set of informers fails on several folds and the zeros f-measures from these failures lower the average results, while the unfiltered informer set never fails and never returns

zeros, and consequently always appear better. As before, a more fair and informative study is to compare the folds that did not fail in the first test to the same folds with the unfiltered set. These are the results we present in Figure 7. It is immediately apparent now that the unfiltered rule base actually leads to classification results that are on average 18% worse than before. It is clear, then, that the final filtering step remains important even when the cluster merging function uses the same methodology. However, the problem that it can filter out all clusters and return a useless empty list remains. In the next subsection, we will study the filtering step in more details and see how we can deal with it.

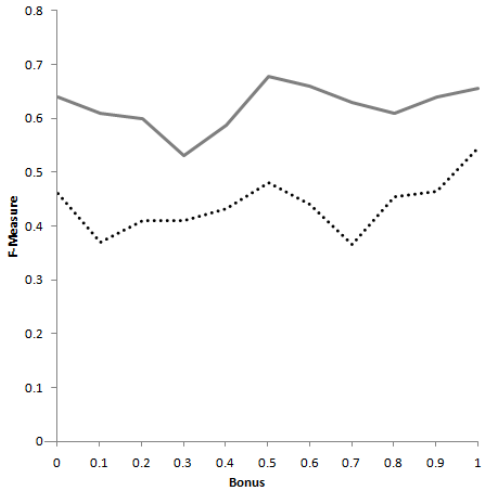

Fig 7: Performance of the second quality evaluation method for different merger bonus values, with (solid grey line) and without (dotted black line) the final filtering step.

4.1.3 Final filtering step

The final step of our algorithm is to filter out non-predictive informers from the rule base. As we explained in Section 3.2, these are informers that have become so general through repeated merging that they no longer model features of a single question type, but could apply equally well to all question types. These informers would at best have no effect, if they are always supplanted by more similar type-specific informers, or at worst lead to misclassifications if they are not.

In our first experiment, we studied the impact of the threshold present in the pseudo-code of Figure 4. This threshold controls the tolerance the system has for informers that cause misclassifications. At the default value of 1.0, an informer must lead to as many or more correct classifications than incorrect ones to be retained. At the other extreme, a threshold of 0.0 retains all informers, even those that do not correctly classify any queries.

In Figure 8, we present the impact of changing the threshold from 0.0 to 1.0 on the average F-value of the five-fold experiment (solid grey line) and on the average final number of informers learned (dashed black line). For this first test, the other options of the system are left at their default settings described in Section 4.1. The figure reveals that the number of informers remains stable in the 120-130 range until the threshold increases to 0.8, at which points it begins dropping sharply. The F-measure seems to have the opposite behavior: it peaks at 0.7 and remains in its highest plateau until 0.9, where it then also drops sharply. It is worth noting the fact that the algorithm failed for one of the folds when the threshold was set to 1.0. Nonetheless, this result indicates that filtering the non-predictive rules from the rule base is beneficial, but within limits; setting the bar too high filters out useful rules, as indicated by the drop and failure at 1.0.

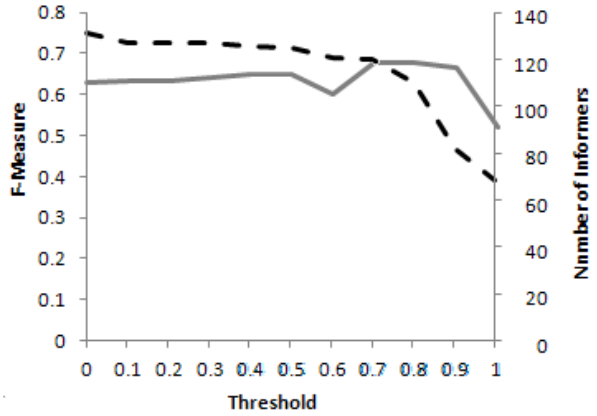

Fig 8: Impact of the threshold impact on the average F-measure (solid grey line) and the average number of informers (dashed black line).

4.1.4 Using initial informers

We noted in Section 3.1 that, in our past work on question-type-informer-learning (Khoury 2011), the system we designed started with a set of 16 simple informers. Including these informers allowed the system to focus its learning process and to converge on a lower number of final informers. It would thus be interesting to study the impact of adding these informers into the current clustering system. The first question to settle, however, is at which point in the algorithm the informers should be inserted. Referring back to the algorithm of Figure 3, there are three points where new informers can be inserted. First, they could be added to the set of training queries received as input by the clustering algorithm, before step 1 of the algorithm. They would then be treated as regular queries and paired off into clusters. The second option is to insert them into the list of initial clusters between steps 2 and 3, before the merging algorithm begins, and to have them treated as regular cluster centers. In this case each of the 16 informers would be a cluster center influencing the merging, by contrast to the first approach where they were necessarily paired off and modified prior to the merging. The third and final option is to insert the new informers into the final set of cluster centers learned, before the final filtering of step 7. This would make it like the system had learned these 16 informers by merging other clusters. We ran the system using each of these three options, and compare the results in Table 2. For comparison, this table also includes the results of the basic system without adding extra informers. The first thing to note when studying Table 2 is that the version of the system without initial informers fails on one fold, while the three versions that add in the basic informers do not. This explains the noticeably lower average F1 value and number of informers of that system. For an

additional benchmark, we also included in the table the average F1 and number of informers of the four folds for which the algorithms do not fail.

Table 2: Impact of inserting informers at different points in the algorithm

Added as:	F1 all folds	F1 common folds	Num. of inf. all folds	Num. of inf. common folds
Not	0.52	0.65	69	86
Training queries	0.67	0.66	113	116
Initial clusters	0.70	0.70	112	112
Final clusters	0.70	0.70	119	119

Table 2 shows that the main difference between including and not including the extra informers is in the total number of informers returned by the system. Indeed, in the first two cases, adding the extra 16 informers before clustering changes the clustering process substantially, leading to new clusters being learned and to 8 more clusters on average. In the next step in these cases, as well as in the third case where the informers are added after clustering, the final filtering step behaves very differently and retains an extra 30 informers – clearly at odds with our stated goal to keep a minimum of informers. The benefit of these extra informers on the average F-measure is minimal, as they only lead to an average 5% improvement in cases where the algorithm does not fail. To be fair, we'll note again that none of the three versions of the system that included the extra informers failed in the one case in our five-fold test where the basic system failed; however, we showed in the previous subsection that this result can also be realized by changing the final filtering threshold, without artificially adding manually-defined informers.

4.2 Comparison

It is interesting to compare the results obtained in this paper with those we presented in [14]. The system we experimented with in that paper stemmed from the same basic idea but used a substantially different framework and implementation. In that version, the system maintained two lists of queries, one for queries that could be correctly classified by a current set of informers and the other for queries that couldn't. Pairs of queries in the second list were merged together to discover general informers that could classify them, and successful informers were added to the set. This system also required the use of an initial set of informers; in [14] we used the basic informers presented in Figure 2.

The system in [14] discovered 11 new informers by merging misclassified queries, for a final total of 27 informers, far below the number of informers generated from cluster centers in this paper. However, there was also on average 37 queries that remained misclassified and for which no informers could be learned, giving a total of 64 informers in the system overall. In classification experiments, the system shows a precision of 0.80 and a recall of 0.65, for an overall F-measure of 0.71. By comparison, the best version of the system we found in our study in section 4.1 is the one that does not use any initial informers, does not filter the informers before clustering, compares clusters by similarity rather than correct classification rate, and uses a bonus of 0.7 for the cluster merging and a threshold between 0.8 and 0.9 for the final filtering step (we set it to 0.85). In classification experiments, a system with that configuration generates 83 informers, and shows a precision of 0.70 and a recall of 0.63, for an overall F-measure of 0.66.

It is difficult to compare the two systems directly, since by their very design they end up with completely different sets of informers. However, we can contrast them statistically based on their results. It appears that increasing the number of informers does not necessarily entail better results. The system in this paper generated almost 20 more informers than before but the precision dropped by 10%. This echoes the analysis presented in [14], in which we added new informers between two experiments and found that the precision also dropped by 10%. Meanwhile, the recall value remained consistent between the two systems, and this fact limited the drop in F-measure.

The decrease in precision implies that the additional informers in the larger set learned by clustering appear to be very similar (i.e. have similar wording) to queries of different types. Part of the problem comes from the fact that the clustering version of the system takes a "filter-out" approach, in the sense that it maintains all clusters into the final set unless they misclassifies test-fold queries, while the system in [14] takes an "add-in" approach and does not include an informer unless it is shown to correctly classify test-fold queries. This explains both why the final set of informers in [14] is smaller and why it shows a better precision. There is thus a clear benefit to this "add-in" approach, and a version of it should be included to enhance our clustering algorithm in future works.

5. Conclusion

In this paper, we presented an exploratory study of a clustering methodology to group together similar sentences based on their part-of-speech syntax and using a weighted part-of-speech hierarchy we designed. We believe that such an algorithm could have numerous applications, including for instance key phrase detection or syntax error correction. In order both to demonstrate the use of this algorithm and to test its performance, we applied it to the task of question type classification, specifically to learning type informers like those suggested in [5]. The results obtained are very promising: different variations of the algorithm cluster together between 75% and 80% of the training queries, and the cluster centers, obtained by merging together all queries in a cluster using our part-of-speech hierarchy, can then be used directly as type informers to classify the queries

with up to 70% f-measure. These results demonstrate the validity of the proposed methodology, and are quite encouraging for a first prototype of a new technique. There are many directions to explore to further improve and refine the clustering algorithm, including both modifications of the current algorithm such as using the "add-in" approach mentioned earlier, and major revisions such as implementing it as a k-nearest-neighbor clustering algorithm. These changes will likely lead to a more robust algorithm and even better results.

References

[1] Suzuki, M., Kuriyama, N., Ito, A., Makino, S. Automatic clustering of part-of-speech for vocabulary divided PLSA language model. International Conf. on Natural Language Processing and Knowledge Engineering, 2008, pp. 1-7.
[2] Chen, M, Song, Y. Summarization of text clustering based vector space model. IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design, 2009, pp.2362-2365.
[3] You, C. H., Lee, K. A., Ma, B., Li, H. Self-Organized Clustering for Feature Mapping in Language Recognition. 6th International Symposium on Chinese Spoken Language Processing, 2008, pp. 1-4.
[4] Khoury, R., Karray, F., Kamel, M. Keyword extraction rules based on a part-of-speech hierarchy. International Journal of Advanced Media and Communication. 2008, 2(2):138—153.
[5] Krishnan, V., Das, S., Chakrabarti, S. Enchanced Answer Type Inference from Questions using Sequential Models. Proceedings of Human Language Technology Conference / Conference on Empirical Methods in Natural Language Processing, 2005, pp. 315—322.
[6] Yin, B., Ambikairajah, E. Chen, F. Improvements on hierarchical language identification based on automatic language clustering. IEEE International Conf. on Acoustics, Speech and Signal Processing, 2008, pp. 4241-4244.
[7] Froud, H. R., Benslimane, A., Lachkar, A., Ouatik, S. A. Stemming and similarity measures for Arabic Documents Clustering. 5th International Symposium on I/V Communications and Mobile Network, 2010, pp. 1-4.
[8] Meedeniya, D.A., Perera, A.S. Evaluation of Partition-Based Text Clustering Techniques to Categorize Indic Language Documents. IEEE International Advance Computing Conference, 2009, pp. 1497-1500.
[9] Razmara, M., Fee, A., Kosseim, L. Concordia University at the TREC 2007 QA track. Proceedings of the Sixteenth Text REtrieval Conference, 2007.
[10] Liang, Z., Lang, Z., Jia-Jun, C. Structure analysis and computation-based Chinese question classification. Sixth International Conference on Advanced Language Processing and Web Information Technology, 2007, pp. 39—44.
[11] Tomuro, N. Question terminology and representation of question type classification. Second International Workshop on Computational Terminology, 2002, vol. 14.
[12] Harabagiu, S., Moldovan, D., Pasca, M., Mihalcea, R., Surdeanu, M., Bunescu, R., Girju, R., Rus, V., Morarescu, P. Falcon: Boosting knowledge for answer engines. Proceedings of the 9th Text REtrieval Conference (TREC-9). 2000, pp. 479–488.
[13] Zhang D., Nunamaker J. F. A Natural language approach to content-based video indexing and retrieval for interactive e-learning. IEEE Transactions on Multimedia. 2004, 6(3):450—458.
[14] Khoury, R. A Learning Algorithm for Question Type Classification. Proceedings of the 2011 International Conference on Artificial Intelligence, 2011, 1:265-371.
[15] Marcus, M., Santorini, B., Marcinkiewicz, M. A. Building a large annotated corpus of English: the Penn Treebank. Computational Linguistics. 1993, 19(2):313—330.
[16] Dang, H. T., Kelly, D., Lin, J. Overview of the TREC 2007 Question Answering Track. Proceedings of the Sixteenth Text REtrieval Conference (TREC 2007), 2007.

Richard Khoury: Assistant Professor in the Department of Software Engineering at Lakehead University. Dr. Khoury's primary area of research is natural language processing, but his research interests also include data mining, knowledge management, and machine learning.