Clustering Automatically Parsed Text Reviews Using Ant Clustering Algorithm

Thomas Bohlken, Floris den Hengst, Rutger van Willigen

Abstract

In this paper we investigate the possibilities to cluster annotated hotel reviews using an ant clustering scheme, which is usable in the recommendation domain. We include a basic ant clustering algorithm and extend it with a cooling-down-scheme for picking up items to boost convergence. Also, we equip the ants with a memory of previously dropped off items and compare different sizes, in order to improve the algorithm's performance. We show the possibility of using an ant clustering algorithm to cluster hotel reviews, but in our analysis we conclude a more suitable dataset should be used to ensure that a valid conclusion on the performance of the algorithm is drawn.

1. Introduction

There are a lot of types of recommendation systems, often based on common types of recommendation algorithms. These algorithms usually take the same type of data as input, such as rating on data items or properties of data. This, however, limits the amount of input as it takes time and effort to collect this type of data, as well as the quality of the data as it is 'compressed' into such a suitable format.

In this paper we try to surpass this hurdle using textual reviews about hotels, a lot of which can be found on the Internet. There are many review sites (e.g. Zoover, TripAdvisor) on which customers can give reviews containing much more detailed information than what can be summarized in categorized ratings. This information, however, is often unsuitable for recommendation systems as it is difficult to extract it correctly.

We search for a way to bridge the gap between these two difficulties and investigate a system that can extrude the type of data needed for an Ant Clustering Algorithm (ACA) from a dataset of natural language hotel reviews, which is not yet suitable for such an algorithm. After the raw data has been parsed and converted in the right configuration, we will use the ACA to see if the data can be clustered in meaningful clusters.

An application of these clusters might be a hotel recommendation system.

2. Experiments

The experiments done consist of three parts. First the dataset, consisting of 1075 reviews of 469 different hotels, is parsed by the KAF (KYOTO Annotation Format) parser. The KAF parser was built and trained to extract sentiment on relevant topics in the hotel domain from these reviews. Second, the parsed data is converted into hotel specific data items which are usable by the ACA. Finally, the ACA will cluster the converted data and the results are analyzed.

We will now discuss each of these steps in more detail.

2.1 Data source and KAF-parser

The hotel reviews were provided by the Faculty of Arts at the Vrije Universiteit Amsterdam. The hotel reviews are extracted from multiple review websites and have a length of approximately 50 to 80 words. We have a set of 1075 reviews, with a total of 469 different hotels (~2,3 reviews per hotel on average).

The KAF-parser can be accessed at [3]. It is built to mine opinions about hotels from plain text files. It recognizes opinions about certain abstract properties such as "restaurant" or "service" and labels them with a polarity (positive or negative) and strength (1 for positive, 2 for stronger positive, -1 for negative, etc). In total, 1102 opinions on properties were extracted. This is an average of only 1,04... opinions per review, which unfortunately means a lot of information was lost in the parsing process.

2.2 Converting KAF to data items

The data items that are used in the ACA represent all represent one of the hotels in the dataset. Each hotel is represented by a summary of the opinions that have been extracted from the reviews using the KAF-parser. Not all opinions have been recorded in the summaries however. This is because of two reasons. First of all, the topics of the opinions are not always relevant for the hotel that is discussed in the review. This is because some of the opinions are about topics that are not directly or not at all related to the hotel. For instance, the expression "*It was sad that I came down with the flu this holiday"* is not relevant to process in the review. Secondly, although the KAF-parser has overall is very potent, there are still some errors found of incorrectly parsed opinions. To overcome this problem only opinions that have topics which also appear in the list of properties are used in the summary of a hotel. The list of properties is as follows, based on the classification in the KAF-parser:

Room	Sleeping comfort	Staff
Facilities	Restaurant	Value for money
Swimming pool	Location	Bathroom
Parking	Noise	Cleanliness
Breakfast	Internet	

The summary of a hotel exists of the average value over all the reviews for a hotel over all existing opinion topics. The topics for which no opinions were present are given the value 0. This represents indifference about this topic. Remember that the KAF-parser was able to extract only about 1 opinion per review, resulting in a large proportion of the data having this 0-value.

2.3 Ant Clustering Algorithm

To implement the ACA, we based the basic algorithm on [1]. However, in order to make a more thorough evaluation of the research question we also implemented to other techniques on top of the basic ACA, namely a cooling schedule for the pickup chance and a technique to implement a simple memory in the ants. These techniques will, respectively, help to let the ACA converge [2] and help let the ants make less clusters [1].

2.3.1 Basic algorithm

The basic ACA, based on [1], exists of a main loop that will randomly select one ant from the colony and let that ant either pick up a data item or let it drop a data item or do nothing at all and then let the ant make a random movement. This loop works as follows:

```
1       while true {
2             for all ants {
3                   if ant carries item i {
4                         ant drops i with probability Pd(i)
5                   } else if an item i is on the same location as ant {
6                         ant picks up i with probability Pp(i)
7                   }
8                   move ant in a random direction with random step size
9             }
10      }
```

In this loop two probabilities are defined; the probability to drop an item *Pd(i)* and the probability to pick up an item *Pp(i)*. These two probabilities are expressed in

$$Pp(i) = (Kp / (Kp + F(i))^2$$

where *Kp* is a constant, and

$$Pd(i) = \begin{array}{ll} \text{if } F(i) < Kd & 2F(i) \\ \text{else} & 1 \end{array}$$

In both function *F(i)* is defined which represents the local similarity. This is a measure to detect how similar a data item is in comparison to the other data items that are in the local view of an ant. This is calculated by the formula

$$F(i) = (1 / d^2) * SUMj(1 - d(i,j) / alpha)$$

where *j* are all data items within the local area of an ant, *d^2* is the total number of sites within this local area(or in other words, the maximum number of data items within the local area), alpha is a constant and *d(i, j)* is the similarity between two data items *i* and *j*. This similarity is problem dependant and in our case will be a simple Euclidian distance calculation represented by

$$d(i, j) = SQRT(SUMi=0 \to n (qi - qp) ^ 2)$$

where *n* is the amount of dimensions in the data.

The main improvements over the original method from [1] are the random step size (line 8) instead of a fixed step size, and the parallel update of all ants in a time step. The random step size improves the relatively low level of randomness of the original method, while the parallel exhibits a more natural behavior of the simulated ants. We thus try to improve one of the AEGIR4 properties of the ant approach for data clustering in our basic algorithm.

2.3.2 Cooling schedule for pickup chance
An extension for the basic algorithm is based on [2], where a cooling schedule for the pickup chance is introduced. This technique entails lowering the pickup chance by some degree after a certain point for each interval, by increasing the pickup threshold for ants to pick up data items over time. The idea behind this is that the algorithm will converge as the number of items being picked up at some point in time will become so low that the configuration of the data items will become stable.

The implementation of this technique is quite simple as it changes only one variable, using a simple mechanism. The only addition that is made is in the main loop, where the following expression is added:

```
if g % Ci == 0 {
      Pt = Pt * Cr
}
```

Where *g* is the current generation, *Ci* is the cooling interval, *Pt* is the pickup threshold and *Cr* is cooling ratio.

Adding a cooling scheme for pickup would provide for a nice chance to introduce more adaptiveness to the ACA. However, since the cooling down function is not enforced through a feedback loop (but rather set manually), the added adaptiveness is small.

2.3.3 Ants with memory
The second adaptation of the basic ACA is somewhat more complex, even though the basic idea is simple. In [1], each ant in the ACA is extended with a memory, in which the last *x* number of picked up data items are stored, along with the coordinates of the drop location. Now, instead of using the local similarity to determine whether or not an item will be dropped, the ant will compare a picked up data item with the data items present in its memory, once the memory has been filled. The data item in the memory that is the most similar with the one picked up by the ant will now become the goal of the ant, after which the ant will walk to the location of the chosen data item and drop the item on that location.

The implementation of this addition consists of changes in two sections of the basic ACA. The first adaptation is made in the calculation of the drop chance. When calculating the drop chance, the goal now also has to be taken into account, as the ant should then only drop the data item when the item is in the local area of the ant. Therefore, when an ant has a goal, the drop chance will either be 0, if the goal is not in the vicinity of the ant, or 1, if it is.

If the ant does not have a goal, the pickup chance is calculated on the traditional way, but only if the memory of the ant has not yet been filled completely. Only after it has been filled, the ant will compare its current load to the data items in the memory of the ant and select the most similar one. In this case the drop chance will be zero as the ant now has a goal.

The second part of the basic ACA that has to adapted is the section that will let the ant make a random movement. This is still relevant, but only when the ant doesn’t have a goal. When the ant does have a goal, it should move towards the location of the goal and not in a random direction as

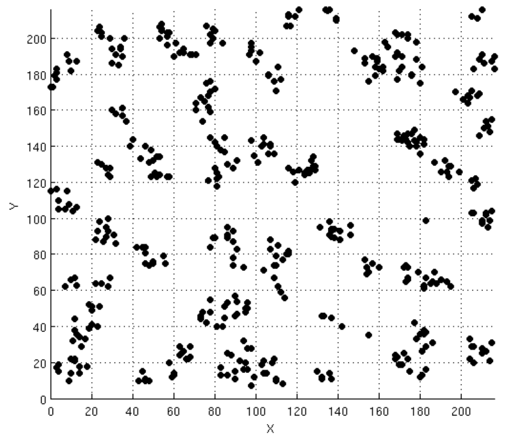

Figure 1: Basic ant clustering algorithm

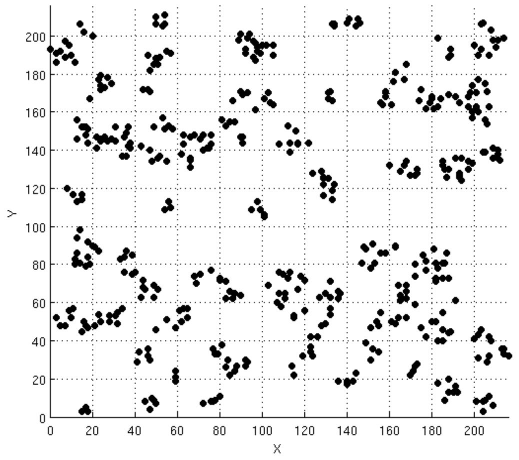

Figure 2: Basic ant clustering algorithm with cooling scheme extension.

this would be very inefficient. Therefore, if the ant does not have a goal a random movement in any direction is made by the ant, otherwise a movement towards to goal is made.

3. Results

The results of the 3 different setups can be found in this chapter. An in-depth analysis is given in chapter 4.

3.1 Basic algorithm

Figure 1 shows the ants at the end of the run 1 (see Appendix A). A dot represents a hotel. As can be seen in the figure, no coherent clusters are formed. Some hotels are placed together, but no real clusters can be detected. This is largely due to the small area the ants cover when moving in a random fashion. Since clustering is so weak, we refrain from further analysis.

3.2 Cooling scheme for pickup chance

Figure 2 shows, again, the ants after run 2 (see Appendix A). Again, little to no clustering can be detected and no actual improvement over the basic algorithm.

3.3 Ants with memory

Figures 3a through 3c show the results after runs of the algorithm with memory implementation.

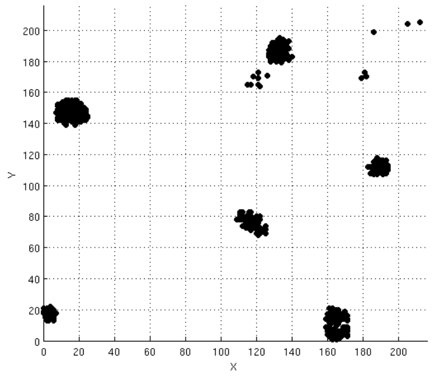

Figure 3a: ACA (memory size 2)

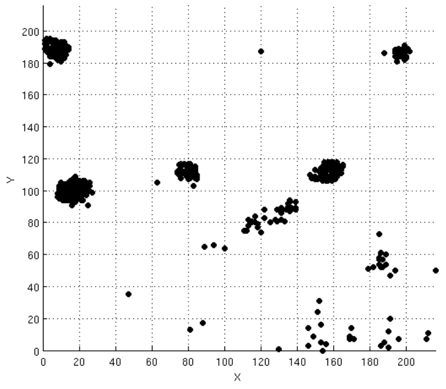

Figure 3b: ACA (memory size 5)

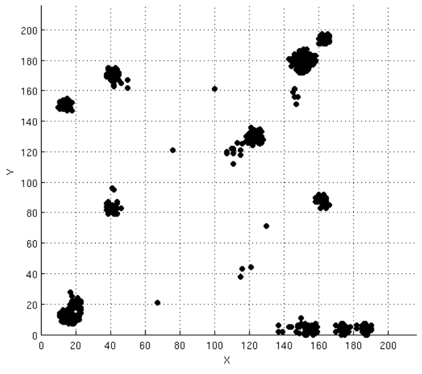

Figure 3c: ACA (memory size 20)

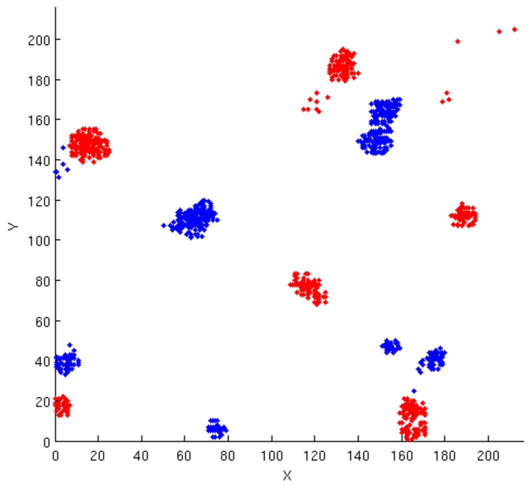

Figure 4a: result of 2 runs of ACA with memory

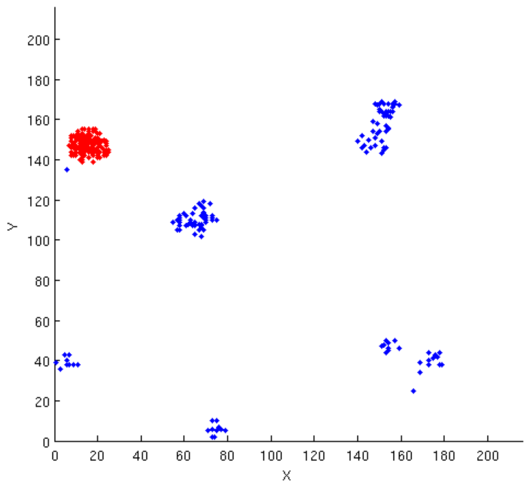

Figure 4b: red clusters data items' location in both runs

Although not all data items are properly assigned to a cluster, we see some clear clusters being formed.

4. Analysis of results

We've shown that no coherent clusters are formed by the first two methods (described in 3.1 and 3.2). The third method (3.3), however, gives the impression of correct clustering of data. We will now further analyze the results for this approach.

In figures 4a and b two images are displayed to analyze the clustering capacity of the ACA in which memory was implemented. The left image shows the final configurations of two runs of the algorithm with memory, one in red and one in blue. On the right, we have isolated one cluster (in red) and show the locations of the data items of this cluster in the second run in blue.

As we can see, the data items that are clustered in the red run are scattered over multiple clusters in the blue run. This shows that although the data seems to cluster properly using the algorithm, the clusters that emerge are not necessarily meaningful.

This outcome could have two different causes; either the clustering algorithm fails to produce meaningful results, or the dataset isn't suitable. We have decided to investigate the dataset further by counting the amount of times each property is mentioned in the entire dataset, consisting of 1075 reviews. The outcome is as follows:

Room	144	Sleeping comfort	52	Staff	278
Facilities	65	Restaurant	246	Value for money	31
Swimming pool	13	Location	150	Bathroom	70
Parking	23	Noise	3	Cleanliness	7
Breakfast	10	Internet	10		

Considering there are only 1,3 opinions per review and only 2,3 reviews per hotel on average we note that most of the data set is made up by the "indifferent" value of 0. This is illustrated by the list above; most properties don't appear often. This means there is a very high similarity between data items. Dropping or picking up an item therefore becomes the result of random factors mainly as a consequence, instead of being based on properties of the data item. Since the similarity between items is used as the main determinant for clustering, the sparseness of the data dearly cripples our algorithm.

5. Discussion

Because of our dataset, it is difficult to test the improvements in our algorithm. We are, however, presenting a couple of possible additions to improve the algorithm's performance in clustering. These are to be tested on datasets with better validity checking possibilities than the hotel review dataset.

In terms of self organisation, adaptiveness is the weakest AEGIR4 property of the ACA. We have tried to improve this property by adding a cooling down function, but as stated before, this function is not triggered by the ants itself and therefore not very adaptive. A possible way to achieve adaptiveness is to let the ants decide to trigger the cooling down function themselves. This could be based on the amount of hotels in the ant's vicinity. If this amount would exceed a certain threshold, this could indicate that a cluster has been formed there.

Another way of introducing adaptiveness could be the introduction of pheromones. In nature, ants drop pheromones which are followed by the others. This model would, however, not be very useful in the ACA as the desired trajectory of an ant should be based on the data items, rather than the ants. Therefore, in [2], it is proposed that data items drop pheromones that decay over time after the item has been moved elsewhere. These pheromone trails influence all the ants' drop and pickup chance when the pheromones are in their scope.

6. Conclusion

In this paper we presented our findings on clustering hotel reviews that were automatically parsed using the KAF-parser. Three different ACA were implemented, namely a basis ACA, an ACA with a cooling down scheme and an ACA with memory implemented in the ants.
The data gathered from all three types of ACA show that the dataset used cannot be properly clustered by any of the algorithms used. The first two do show some kind of ordening in the data, but no real clusters. The third algorithm does show proper clusters, but when we investigated these further, they did not seem to have any sort of meaning.
The fact that no proper clusters were made is most probably due to the dataset. Based on both the size and the quality of the dataset, we can see that the dataset is not usable for a clustering algorithm. To improve the quality of clustering it is either needed that the size of the dataset is increased or that the number of opinions about each hotel is increased.

Bibliography

[1] Lumer E., Faieta B., *"Diversity and Adaptation in Populations of Clustering Ants",* 1994
[2] Vizine A., de Castro L., Hruschka E., Gudwin R., *"Towards Improving Clustering Ants: An Adaptive Ant Clustering Algorithm",* 2005
[3] http://ic.vupr.nl:9081/ner-prop-opin

Appendix A

Experiment setup:

Alpha: 10
dropThreshold: 0.02
pickupThreshold: 0.005
dropConst: 1
pickupConst: 0.01
All ants updated at the same time: True
Generations used: 30000
Memory size: 0
Step size: 3