Opinion Summarization for Hotel Reviews

Bogdan Cristian Marchis
University Politehnica of Bucharest
313 Splaiul Independentei, Romania
bogdan.marchis@gmail.com

Alexandru Tifrea
University Politehnica of Bucharest
313 Splaiul Independentei, Romania
alex.tifrea93@gmail.com

Mihai-Cristian Volmer
University Politehnica of Bucharest
313 Splaiul Independentei, Romania
mihaivolmer@gmail.com

Traian Rebedea
University Politehnica of Bucharest
313 Splaiul Independentei, Romania
traian.rebedea@cs.pub.ro

ABSTRACT
This paper presents a new approach for finding the best n-grams that efficiently summarize a large set of reviews. The proposed unsupervised method uses a readability score and a representativeness score to select those n-grams that best convey the main opinions contained in the processed reviews. In order to further refine the selected n-grams, we use sentiment analysis and part of speech (POS) tagging to impose certain requirements that the n-grams that we are looking for should meet. Furthermore, the best n-grams were classified into several topics, which allowed a better prevention of redundancy among the summarizing n-grams. Therefore we offer an unsupervised, mostly non-aspect based, unstructured opinion summarization algorithm that can be easily implemented for any web platform that accepts reviews, due to its genericity. In order to assess the results of our algorithm, we summarized hotel reviews extracted for the TripAdvisor[1] website. The algorithm produces readable results that convey relevant opinions about the hotels that we used for testing.

Author Keywords
Opinion summarization, opinion mining, natural language processing, n-grams, micropinions.

ACM Classification Keywords
H.5.2. User Interfaces: Natural language, I.2.7 Natural Language Processing.

INTRODUCTION

Almost every online platform that offers services to customers has a reviews section nowadays. This is one of the building blocks on which e-Commerce relies, but it also raises a few problems. The main issue is that popular services like Amazon, eBay or TripAdvisor can end up having hundreds or thousands of reviews for a single product. Of course, this makes it very hard for users to go through all the reviews and extract the relevant opinions from all of them. Another problem, which makes it all the more difficult for users to form an opinion about a product, is the fact that quite a few of the reviews do not talk about a concrete aspect of the given product, but rather use very general phrases to describe it, such as '*The hotel was great*' or '*I didn't like it at all*'.

Generating an opinion summary starting from a very large text is something that has been extensively researched in the last few years [1, 7, 8, 9, 11, 12] as the number of opinions on the web is always increasing, thus making it harder and harder for users to take into account every opinion when deciding upon a product, a service or, in our case, a hotel. Opinion summarization can vary from simply giving an overall rating [14] based on all the ratings for a single hotel, or generating a rating based on what each user had to say about that hotel, to generating a new review that will be a summary of every other review [2, 4, 6, 17]. Generating an opinion summary can be compared to normal text summarization - finding a small set of keywords or key phrases in the text that best describe the overall text [10] - with the difference that not all keywords or key phrases can be considered an opinion. This makes opinion summarization more difficult than generic text summarization.

In this project we seek to generate a few sentences that best represent a summary for hundreds or thousands of reviews. In order to do this, we thought that the best approach was not to use the sentences that were already in the reviews, but to create new sentences, based on the text, and then check how representative each sentence is for the entire set of reviews. We chose this approach because it is often the case that a sentence covers more than just one topic and contains more than one opinion. By creating new sentences instead of extracting sentences from the reviews, we no longer have to make sure the sentence that we selected from the original text is appropriate (i.e. the sentence that exactly matches the opinion that the algorithm decides to output). Generating a new sentence is a demanding task, not only because the sentence has to be grammatically correct, but also because it has to convey an opinion that is relevant to the large set of reviews. Moreover, with such an approach, that only uses a representativeness score to filter opinions, it is very hard to keep among the selected n-grams those which contain negative opinions, since these are, based on our observations, far less common than the positive ones.

[1] www.tripadvisor.com

We take this aspect into account and offer certain bonuses in order to keep a good balance between positive and negative reviews, regardless of what the general opinion about the given hotel is. For instance, a generally well-seen hotel, can still have flaws. This is why we want to make sure the reviews mentioning these flaws stand a chance in the face of the more numerous positive reviews.

Our approach is mostly non-aspect based, meaning that we don't look for opinions related to particular aspects of the product/hotel. All the resulting n-grams that summarize the reviews are built and selected only based on their readability and representativeness scores. We also give bonuses to n-grams that are formed around certain syntactical structures as will be explained in the following sections. It is only after all the n-grams are computed and sorted by their scores that we use some predefined topics to extract only those summaries that are relevant to our product. This is the aspect based part of the algorithm.

RELATED WORK

State-of-the-art algorithms for review summarization are usually aspect based, looking for opinions related to certain features of the product that is reviewed [8]. This approach is highly dependent on the way the features are chosen. Syntactic tree parsing, POS (part of speech) tagging or a supervised approach are just some of the methods that are very common when trying to extract the features that need to be analyzed.

In generic text summarization, most attempts use extractive summarization which consists of extracting relevant and representative fragments from the given text. Some important contributions in this direction are [7, 11, 12]. Abstractive summarization is considerably harder than the extractive approach, since it raises the problem of generating readable phrases that can also convey a relevant opinion, which is also representative for opinion summarization.

Our work builds on the results obtained by [1]. They propose an abstractive unsupervised algorithm that extracts micropinions and generates short sentences to present them to the users. Unlike their work, ours is a hybrid approach, using a non-aspect based algorithm to generate the candidate sentences and then using some predefined topics (i.e. features) to select only sentences relevant to the product which is being reviewed. In addition to that, we use a slightly different algorithm for generating the candidates, which allowed us to use some tools more efficiently (i.e. Microsoft Ngram Service, CoreNLP etc.) by traversing the solution space in a breadth-first fashion. Moreover, our version of the algorithm lends itself to working with an adaptive pruning mechanism, which allows us to select at each step only valuable n-grams. Thus we can improve the efficiency of the algorithm up to the level that we desire by heuristically adjusting the thresholds used for the readability and representativeness scores. Of course, this has a direct impact on the quality of the results, but can prove useful when running an implementation for a system with limited resources, like a mobile device.

Another work that is similar to ours is [9]. They offer both an abstractive and an extractive algorithm. However, in their abstractive approach, they generate whole paragraphs, using complex natural language processing tools in order to achieve this. Rather than doing this, we aim to create small, concise sentences, not longer than 10-15 words.

OPINION SUMMARIZATION SYSTEM

We wanted to create an algorithm that would find a few sentences that would best summarize the opinions of a few hundred people. In our opinion, the best solution was to find short sentences, between three and eight words, that would summarize the set of reviews as good as possible. The main reasons for selecting this approach are the following. Firstly, it is fairly easy to construct short sentences based on the initial text and it also means that it will be easy to check the sentences for their readability and their representativeness, which we are going to discuss a bit later. Secondly, having a short sentence also means that it is going to be easier to compare it with another sentence, which helps us reduce the run-time considerably.

In order to generate the sentences that best summarize the opinions of hundreds of users, we decided that the best solution would be to use a bottom-up approach and create new sentences based on the words from our initial text (which is made by concatenating all the reviews of a given hotel). Although we are using words from the initial text, the words in the newly created sentences do not have to be in the same order as they were in the original sentence, making sure that we have a better chance of creating a sentence that summarizes the opinion of more than one user. If we consider a sentence to be represented as s_i, our final result should be a set of sentences, $R=\{s_i\}_{i=1}^{k}$ and each word from s_i should be a word from our original text $T=\{s_i\}_{i=1}^{n}$, which is also a set of sentences.

Since we earlier stated that the sentences we create might be different from every sentence in the original text, we must create some functions that will assure us that the result is readable and is relevant to our initial set of reviews. The first functions that we need to take into consideration are a readability score [5], which tells us how readable a sentence or a small group of words is, and a representativeness score, which tells us if the sentence is relevant for our initial set of reviews. After creating these two functions, our job should only be to determine which sentence has the best score of representativeness and of readability:

$$R = max_{R=\{s_1,s_2,\ldots,s_k\}} \sum_{i=1}^{k} (S_{rep}(s_i) + S_{read}(s_i))$$

where:

1. S_{rep} is a function for scoring the representativeness of a sentence
2. S_{read} is a function for scoring the readability of a sentence

Once we have a set of sentences, R, we have to take into account that two or more of those sentences might have the same meaning. To address this problem, as we mentioned in the paragraph above, we have to define a similarity function, S_{sim} which will give us the similarity score between two sentences. If the result returned by the similarity function, $S_{sim}(s_i, s_j)$, is above a similarity threshold, which was determined heuristically, we will consider the two sentences, s_i and s_j, to be similar. If we find that two sentences are similar, we have to eliminate the one that yields the lowest result for the formula: $S_{read} + S_{rep}$.

Once we have these functions, we can determine the best sentences by using a threshold for each function and a threshold for the sum of the representativeness function and the readability function:

$$R = \{s_i\} \text{ for } i = 1 : k \text{ where } S_{rep}(s_i) \geq \sigma_{rep}, S_{read}(s_i) \geq \sigma_{read} \text{ and } S_{sim}(s_i, s_j) \leq \sigma_{sim} \forall s_i, s_j \in [1, k]$$

where:

- σ_{read} is the minimum readability accepted for a sentence
- σ_{rep} is the minimum representativeness accepted for a sentence
- σ_{sim} is the maximum similarity between two sentences

Readability

The purpose of the readability function is to tell us if a sentence is grammatically correct and to make sure that the sentence makes sense for a user. This function is especially important in our case, since we are creating new sentences by combining every possible word from our initial text. At this point, we have to realize that when we put together two words from a text, most of the results will make no sense at all. Since most of the results will make no sense, we can drastically reduce the number of word tuples by setting a high enough σ_{rep} value.

In our implementation of the readability function, we chose to use the Microsoft Web N-gram Service. This cloud-based platform provides a joint probability score for a given sentence. The cloud platform provides a readability score for a sentence and uses as training set all documents indexed by Bing in the en-us market.

To access this service we used the Python module provided on the Microsoft N-gram blog from MSDN. On top of the module we implemented a server-like interface. Thus, the Python module runs in parallel with the main process. Queries are sent from the main process to the Python module, which resolves them with the Microsoft N-gram server.

Representativeness

The representativeness function determines whether a newly created sentence is found in the original reviews, how many times it is found and in which combination of words. We will not take into account how readable a sentence is when computing its representativeness score, because for this part we only care if a sentence appears in a form or another in the original text.

The first thing considered when computing the representativeness score is whether or not the words that form the new sentence are actually closely related in the original text. In order to make sure the words are related in the original text, we first check whether the words are in the same sentence, and if so, how many times they appear in the same sentence and whether the distance between the two words, inside the sentence, is not greater than a window size, C. The first condition makes sure that two words that have a high appearance rate in the initial text, but almost never appear together, will not be able to form a new sentence. The second condition makes sure that strongly connected words have a better chance of forming a new sentence, then words that just happen to appear in the same sentences a lot.

We will define S_{rep} as:

$$S_{rep}(s_i) = S_{rep}(w_1 w_2 ... w_n) = \frac{1}{n} \sum_{j=1}^{n} pmi_{local}(w_j)$$

Where pmi_{local} is defined as the local pointwise mutual information (PMI) function:

$$pmi_{local} = \frac{1}{2C} \sum_{k=j-C}^{j+C} \log_2 \frac{p(w_j, w_k) \cdot c(w_j, w_k)}{p(w_j) \cdot p(w_k)}$$

Where:

- C is the size of the window within which we look for closely related words
- $p(w_j, w_k)$ is the frequency of two words co-occurring in the same sentence
- $c(w_j, w_k)$ is the frequency of two words co-occurring in the same sentence and in the same window
- $p(w_j)$ is the frequency of a word in the text

Let us consider the following example: if we have a text made of twenty sentences and two words, w_j and w_k, which appear in the text ten times each. The words also appear five times in the same sentence and twice in the same window, C. The values for the relations above, will be $p(w_j, w_k) = 5/20$, $c(w_j, w_k) = 2/20$ and $p(w_j) = p(w_k) = 10/20$, which means that we get the final value: $pmi_{local} = 1/(2C) * \log_2(0.1)$.

In order to speed up the implementation of the system, a hash table has been used, with the word as a key, and a vector as a value, in which we can find the exact position of every word. Using this structure, we know exactly in which sentence we can find a word, so we only need to iterate through that sentence when computing the pmi. This structure gives us a worst case time complexity of O(n*m*l) where n is the total number of unique words, m is the number of sentences in the initial text and l is the maximum size of a sentence, in words.

The representativeness value is computed for each n-gram before the readability score because the readability score is much more time consuming, since the score is not computed locally, but rather on a remote web server. By computing the representativeness first, more than three quarters of the original n-grams were eliminated, since most of the n-grams are not strongly related.

Similarity

Looking for the best sentences to represent the initial text we noticed that it might be the case that most of the sentences have the same meaning. This might happen if every user says something like '*The hotel was very clean*', but in many different ways. In order to prevent this from happening, we need a good similarity function. We use the Jaccard index to compare two n-grams:

$$J = \frac{|w_i \cap w_j|}{|w_i \cup w_j|}$$

Of course, this only has to do with the sets of words of the n-grams, so no semantic analysis is performed (this may be seen as possible further work). At all times, the queue containing n-grams waiting to be processed only contains n-grams that are not similar to one another.

It is rather important what course of action is taken when a new n-gram that is about to be added to the queue does not pass the similarity test. There are two different scenarios that are considered:

i) the new n-gram is similar to just one n-gram in the queue; in this case, we simply compare the readability and representativeness scores of the two n-grams and only keep the one with the highest scores;

e.g.: the queue is [..., *bathrooms are ok*, …] and we try to add *bathrooms are nice*

ii) the new n-gram is similar to more than just one n-gram in the queue; this means that some part of the new n-gram is similar to one existing n-gram, and some other parts are similar to other n-grams (because otherwise it would mean that the same fragment of the new n-gram is similar to all the existing n-grams with which it doesn't pass the similarity test which leads to a contradiction since we stated before that all none of the n-grams in the queue are similar to one another). When this happens it usually means that the new n-gram conveys information about more than just one topic (since it overlaps with two existing standalone n-grams). In this case, we drop the new n-gram.

e.g.: we try to add *bathrooms are nice and staff is helpful to the queue*: [..., *bathrooms are nice*, *staff is helpful*, ...]

General algorithm

As stated earlier, the easiest way to generate sentences that would best summarize the reviews is to generate the sentences using words from the initial text. If we generate new n-grams based on the words that are in the original reviews, we have a better chance of generating sentences that better summarize the opinion, than we would have if we were to start from scratch.

The first step of the algorithm is to look for the most frequent unique words in the text. The final n-grams generated by the system will only have words that have a frequency higher than a threshold, σ_{freq}. This threshold was determined heuristically and is usually dependent on the number of reviews that are in the dataset. Once we have the most frequent words, we use backtracking to generate every possible bi-gram. However, we employ the pruning algorithm described in the previous section, which eliminates every bigram that has no chance of existing in the original text. We also have to mention that we check every possible permutation of a bigram, hence, we will compute the representativeness score only once, but we will determine which permutation is the best by using the readability score. There is no point in checking the similarity at this point since we only have bigrams and we will only use them as a seed to generate new n-grams. In order to make the algorithm faster, these bigrams are stored in a hash table, for which the key is the first word of the bigram, so that it will be easy to add the bigram to an existing n-gram.

The values for the thresholds that we used in order to select only those n-grams that meet certain requirements (e.g. high readability score, high representativeness score, limit redundancy using the similarity score etc.) were chosen heuristically, by considering both the execution time of the algorithm and the quality of the resulting summaries. The window size C used in the representativeness formula was also chosen by testing different values for it, between 5 and 15.

In the process of generating (n+1)-grams we use the n-grams that we already have at this stage and try to concatenate them with every possible bigram that we have not eliminated after the first stage of the algorithm. A new (n+1)-gram can be formed only if the last word of the n-gram matches the first word of the bigram. We use Breadth-First Search (BFS) to generate the (n+1)-grams since the requests for the Microsoft Web N-gram Service are the most time consuming parts of the algorithm. By using BFS, we can generate every possible (n+1)-gram, eliminate the

worst ones using the representativeness score and only afterwards interrogate the cloud-based platform.

Generate (n+1)-grams
1: Input: queue in which we have every n-gram
2: Output: new_queue in which we will have the (n+1)-grams
3: FOR every n-gram in queue
4: FOR every bigram matching n-gram
5: new_ngram ← JoinNgrams(n-gram, bigram)
6: if new_ngram.representativeness > σ_{rep}
7: new_queue.push(new_ngram)
8: ENDIF
9: ENDFOR
10: ENDFOR
11: GetReadability(new_queue)
12: FOR every new_ngram in new_queue
13: IF new_ngram.readability > σ_{rep}
14: FOR every ngram in all_ngrams
15: IF GetSimilarity(new_ngram, ngram) > σ_{sim}
16: eliminate the worst ngram
17: ENDIF
18: IF new_ngram is not similar with any other existing ngram
19: all_ngrams.push(new_ngram)
20: ENDIF
21: ENDFOR
22: ELSE
23: new_queue.delete(new_ngram)
24: ENDIF
25: ENDFOR

If the similarity function finds that two n-grams are similar, we are going to keep only the best n-gram, which will be the one with the highest sum between the readability score and the representativeness score. We could check the similarity only at the end of the algorithm but we found that the number of similar n-grams generated this way was quite large, so a lot of time is wasted to generate and assess new n-grams that were just (partial) copies of other n-grams. Once we have the (n+1)-grams in *new_queue*, the algorithm proceeds recursively to generate new n-grams until we reach the maximum size of an n-gram, eight, or until none of the newly created n-grams pass the pruning stage.

Sentiment Analysis

The best way to summarize the opinion of a user is to capture his or her sentiment [13, 15] towards a hotel. Since the purpose of the algorithm is to summarize how good or how bad a hotel is, the stronger the sentiment expressed by a user is, the better. The tool that we found fit to analyze the sentiment of a sentence was CoreNLP [3]. CoreNLP provides a rating for a given sentence, using the following classes: very negative, negative, neutral, positive and very positive. Based on the rating that is returned by CoreNLP, a bonus is added to each n-gram. The bonus added for an n-gram that has a very positive or a very negative rating is higher that the bonus added for an n-gram rated as positive or negative.

After early testing, we found that most of the sentences that we were creating had small grammar problems. The sentences were readable, but the words inside the sentence were not quite in the correct order. For example we would get '*nice the staff was*' instead of '*the staff was nice*'. The first thing we did was to find the readability of every permutation possible for the initial n-gram and choose the one with the best readability. The second solution was to rearrange the words inside the sentence until the words are in the write order. To change the order of the words inside a sentence, we only need the part of speech for each word, which we could find by using CoreNLP. Some of the rules that we were using to rearrange the words were: adjective before noun and adverb before verb.

We realized that we could use the fact that we already have the part of speech of each word to improve the generation of new sentences/n-grams. There is no way that we could generate a sentence that would express a strong sentiment without it having at least a noun and an adjective. We thought that we could speed up the generating phase of the algorithm by adding a bonus if an n-gram had a noun or an adjective. We also figured it would be a good idea to penalize a sentence if it contains an interjection, a determiner, a preposition or a number.

Topic detection

After generating every possible sentence and choosing the best ones using the algorithm described above, we have to choose the sentences that are most relevant for each hotel. The best way to determine if a sentence is relevant for a given domain is to find the topic of a sentence and match it with some predefined topics relevant for that domain. In this case, the selected topics were: kitchen, room, staff, noise, location and price.

The tool we used is Word2Vec[2], which is an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. This tool provides an algorithm that, for a given word, creates a list of words having the highest cosine distance related to it. Based on that algorithm, we were able to obtain the cosine distance of two given words.

We created lists of words for each predefined topic. To obtain the topic of an n-gram, we calculate its cosine distance towards each topic. The topic with the highest cosine distance towards the n-gram is the matching one. However, if that distance is not above a minimum threshold, the n-gram is considered to be out of topic and can be dropped from the final output. We established the value for the threshold heuristically, by evaluating the

[2] https://code.google.com/p/word2vec/

performance of the algorithm for several different values, the same way we did with the other thresholds mentioned throughout this paper.

RESULTS

For testing purposes, various datasets for reviews were used. These were collected from the TripAdvisor website and there was a focus on hotels with at least several hundred different reviews. The average number of requests sent to the Microsoft N-gram Web Service is 195 per review. We make an average of 55000 requests per hotel. Obviously, the server requests are one of the most time consuming part of the application. Hence, we limited the number of bigrams at 5000. The number of n-grams with a size greater than 2 is limited at 1500. This limit was never reached, since the number of (n+1)-grams is usually considerably lower that the number of n-grams. This decrease is caused by the pruning of the n-grams which takes place during the BFS traversing.

As stated before, each n-gram is given a score based on readability, representativeness and POS bonus. As expected, during the course of each run, the mean readability value decreases and the mean representativeness value rises as the n-gram size increases.

The readability and representativeness values are constant throughout every output and are not affected by the number of reviews, as seen in Figure 1 and Figure 2. The spike is caused by a hotel that has poorly written reviews. As a consequence, the generated n-grams have lower representativeness score. This forces the algorithm to finish in an early stage.

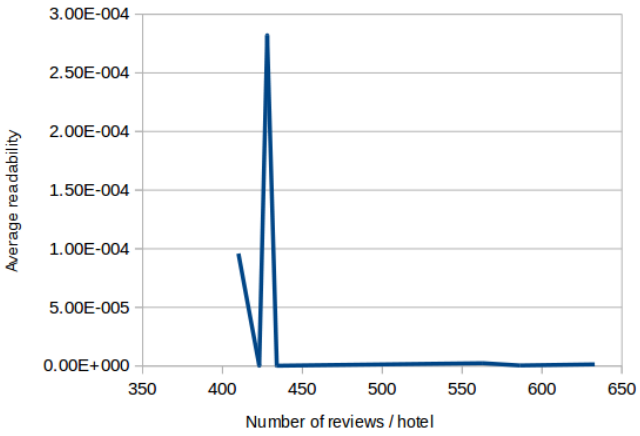

Figure 1: Average readability of the summaries from multiple hotels

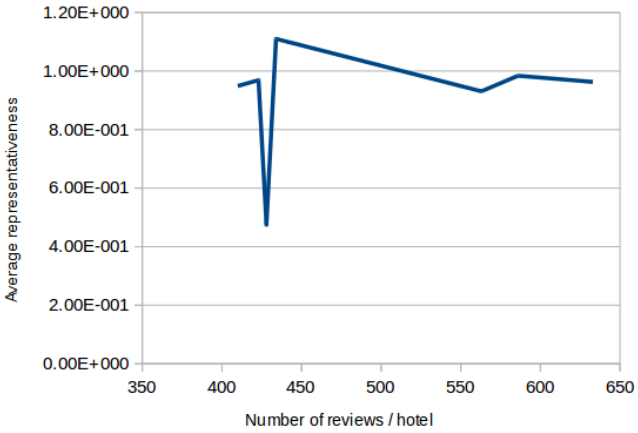

Figure 2: Average representativeness of the summaries from multiple hotels

In what follows, there are some output examples from two different hotels. The application generates for each hotel, on average, 20 summaries. The output is formatted as follows: [<n-gram>, <readability>, <representativeness>, <POS bonus>, <sentiment polarity>] <topic>

Output for Hotel Christina:

[romanian red glass wine and welcome cheese, 1.44092e-09, 1.0253, 0.11 NEUTRAL] food
[bucharest at the hotel i had, 9.28771e-08, 1.09938, -0.092 NEUTRAL] none
[metro bus 5 minutes walk, 4.1583e-07, 0.830261, -0.101 NEUTRAL] location
[accommodating staff very, 8.7571e-10, 0.763112, -0.091 NEUTRAL] staff
[smooth cab she said, 6.06846e-07, 0.528349, 0.11 NEUTRAL] location
[food was excellent, 0.000520357, 0.52147, 0.11 POSITIVE] food
[helpful staff, 7.7023e-06, 0.440511, 0.11 NEUTRAL] staff
[street noisy noise, 4.23309e-05, 0.419229, 0.11 NEUTRAL] noise

Output for hotel Palm Opera:

[will definitely book again soon next, 7.45793e-09, 1.23048, 0.109 POSITIVE] none
[welcoming attentive staff and professional at rest, 9.25606e-09, 1.17844, 0.109 POSITIVE] staff
[location we were good, 2.90115e-07, 1.07646, 0.109 NEUTRAL] location
[available water supply area a lounge touch, 2.2848e-09, 1.07143, 0.109 NEUTRAL] room
[not disappointed me fortunately my wife, 8.25395e-08, 1.03246, 0.11 NEGATIVE] none
[free lounge drinks they offer, 5.15874e-08, 1.02223, 0.108 NEUTRAL] food
[floor bath was lovely had got, 8.14061e-10, 1.0202, 0.109 POSITIVE] room
[children just loved ikea play kitchen, 8.23065e-10, 1.21441, -0.09 POSITIVE] food

In order to assess the quality of the resulting n-grams (e.g. the n-grams that are shown above) we created a survey where summaries could be rated on scale from 1 to 5 (5 being the highest). The survey consisted of 187 summaries from 7 hotels and was taken by six fellow students coming from different backgrounds, namely not just computer science. We have instructed them to focus on how useful a summary is and not necessarily on whether or not it is grammatically correct. After aggregating the data we have obtained an average of 3.12. We have also noticed that this score fluctuates in a rather narrow window of just 1 (i.e. between 2.5 and 3.5) when we vary the number of reviews that we process, as seen in Figure 3.

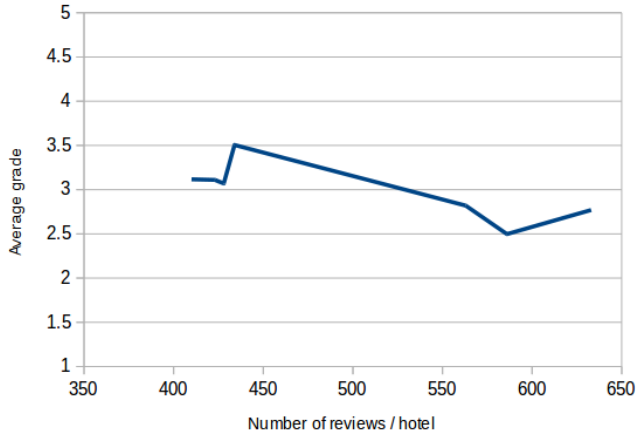

Figure 3: Average grade for hotels with different number of reviews

CONCLUSION

To sum up, our approach offers a fast solution to the problem of finding a set of n-grams that can summarize a given set of reviews without losing the recurrent aspects that appear throughout the reviews. Using the readability and representativeness scores as well as other useful metadata like the POS tags and the sentiment conveyed by an n-gram, the proposed algorithm attempts to use as much information as possible in order to select only the most suitable n-grams. On top of all this, the topic selection functionality makes sure to get rid of n-grams that do not cover a relevant subject as far as the reviewed product is concerned.

As far as other similar attempts go, our approach provides simple, unsupervised and unstructured way to summarize reviews while taking advantage of as much of the available information as possible. Since most of the algorithm is non-aspect based (except for the final part where we start looking for reviews with high scores that also fit a certain topic) it can be changed rather easily to work on different web platforms and different products, even though our implementation uses hotel reviews crawled from www.tripadvisor.com. As for further work, more optimization, including parallelization of some of the operations, will allow for more data to be analyzed in a decent amount of time and, therefore, will greatly increase the quality of the yielded n-grams.

ACKNOWLEDGEMENTS

This work has been partly funded by the Sectorial Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/ 132397.

REFERENCES

1. Carenini, G. and Cheung, J. C. K. Extractive vs. nlg-based abstractive summarization of evaluative text: the effect of corpus controversiality. In *Proc. of the Fifth International Natural Language Generation Conference 2008*, (2008), 33-41.
2. Di Fabbrizio, G., Stent, A. and Gaizauskas, R. A Hybrid Approach to Multi-document Summarization of Opinions in Reviews. In *Proc. The 25th International Conference on Computational Linguistics 2014*, 1 (2014), 23-29.
3. Flesch, R. A New Readability Yardstick. *Journal of Applied Psychology 32*, 1 (1948), 221-233.
4. Ganesan, K., Zhai, C and Viegas, E. Micropinion Generation: An Unsupervised Approach to Generating Ultra-Concise Summaries of Opinions. In *Proc. of the 21st international conference on World Wide Web 1*, 1 (2012), 869-878.
5. Hu, M. and Liu, B. Mining and summarizing customer reviews. In *Proc. of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, (2004), 168-177.
6. Kim, H. D., Ganesan, K., Sondhi, P. and Zhai, C. Comprehensive Review of Opinion Summarization. Technical Report University of Illinois at Urbana-Champaign (2011).
7. Kupiec, J., Pedersen, J. and Chen, F. A trainable document summarizer. In *Proc. of the 18th annual international ACM SIGIR conference on Research and development in information retrieval 1995*, (1995), 68-73.
8. Lin, C.-Y. and Hovy, E. A trainable document summarizer. In *Proc. of the 18th conference on Computational linguistics 2000*, (2000), 495-501.
9. Liu, J., Cao, Y., Lin, C., Huang, Y. and Zhou, M. Low-Quality Product Review Detection in Opinion Summarization. In *Proc. of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning 2007*, (2007), 334-342.
10. Lu, Y., Zhai, C. and Sundaresan, N. Rated aspect summarization of short comments. In *Proc. of the 18th*

International World Wide Web Conference 2009, (2009), 131-140.

11. Manning, Christopher D., Surdeanu, Mihai, Bauer, John, Finkel, Jenny, Bethard, Steven J., and McClosky, David. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proc. of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations 2014*, (2014)
12. Paice, C. Another stemmer. *ACM SIGIR 24*, 3 (1990), 56-61.
13. Pang, B. and Lee, L. Opinion Mining and Sentiment Analysis. *Foundations and Trends® in Information Retrieval 2*, 1–2 (2008), 1-135.
14. Pang, B. and Lee, L. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proc. of the First Workshop on Graph Based Methods for Natural Language Processing 2006*, (2006), 45-52.
15. Pang, B., Lee, L. and Vaithyanathan, S. Thumbs up? Sentiment classification using machine learning techniques. In *Proc. of the ACL-02 conference on Empirical methods in natural language processing 10, 1-2* (2002), 79-86
16. Velikovich, L, Blair-Goldensohn, S, Hannan, K. and McDonald, R. The viability of web-derived polarity lexicons. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics 2010*, (2010), 777-785.
17. Wang, H., Lu, Y. and Zhai, C. Latent aspect rating analysis on review text data: a rating regression approach. In *Proc. of Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining 2010*, (2010), 783-792.