University of Massachusetts Amherst

Concept/Theme Roll-Up

Submitted By
Tanvi Sahay, Ramteja Tadishetti, Ankita Mehta, Shruti Jadon

In collaboration with Lexalytics

In fulfillment of the Data Science Industry Mentorship Independent Study

May, 2017

1 Introduction

The key ideas summarizing a set of sentences can be succinctly represented with the help of short key-phrases that provide information about the theme(s) present in the targeted text. These key-phrases can be used as a means to provide users with information regarding the primary themes present in a set of documents and grouping similar phrases pertaining to the same theme together can allow the users to query the theme that they are interested in, without acknowledging the other topics. This is of particular relevance in the field of hospitality, for both providers and consumers. For example, a hotel owner may be interested in knowing all the bad reviews that their hotel's staff has received to improve the quality of housekeeping and all the good reviews that their food has received in order to improve their menu. Similarly, a customer may only be interested in the kind of facilities the hotel has to offer and not the location or food. In such cases, the hotel owner should have access to all reviews pertaining only to staff/food while the customer should be able to view all reviews pertaining to facilities and filter out the others that she is not interested in. However, since key-phrase extraction techniques do not preserve the semantics or context of the phrase, grouping similar phrases can be a challenging task. In this report, we summarize the methods explored by our team in order to successfully group similar phrases pertaining to a single theme together. For all the methods explained, by similarity, we mean phrases that may be worded differently but occur in similar contexts and share an overlaying theme. For example, given the phrases 'cheese omlette', 'breakfast buffet', 'short ribs' and 'evening party', we try to cluster the first three together into a single group and the last one into a separate group.

2 Pipeline

We divided the task of theme roll up into three prime subtasks:-

1. Data Collection and Key Phrase Extraction
2. Conversion of phrases to their distributed real-valued representation
3. Clustering of similar phrases

A key-phrase, as mentioned above, is nothing but an N-gram that contains information regarding an important theme of a set of sentences. Any N-gram can be characterized by the words contained in the N-gram and the words around it, or the N-gram's context. Keeping this in mind, we explored three categories of distributed representations - using only the words contained in the phrase, using only the context of the phrase and using both the words and the context of the phrase. For clustering, we experimented with two techniques: K-Means Clustering and Gaussian Clustering. The final pipeline has been shown below:
The raw text had to be preprocessed before phrases could be extracted from it. Hence, after data collection, data cleaning was also performed. The next sections describe each part of the above pipeline in further detail.

3 Data Collection

We experimented with two datasets, differing in both size and content. The first dataset we chose was a more general one, which contained letter, articles and media clippings about different government agencies and different types of bills. The data was open

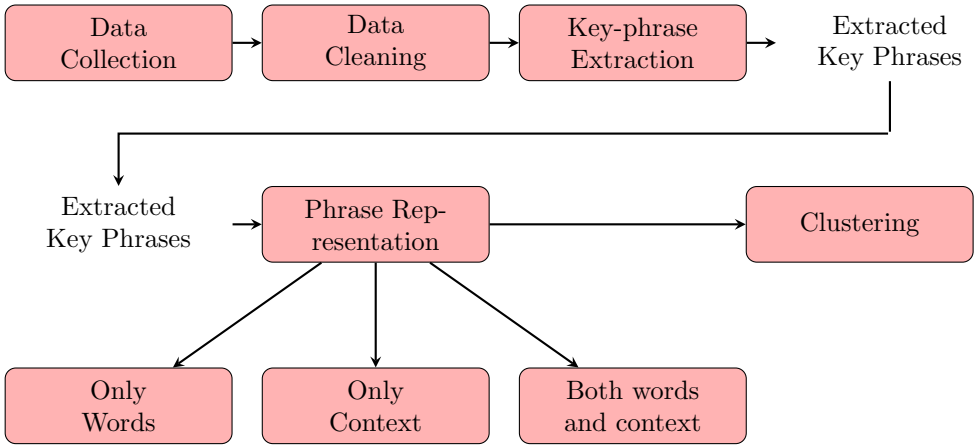

Figure 1: Project Pipeline

source and was obtained from Open American National Corpus and had a total of 286 files of varying sizes. The total token count for this dataset was 1059362 and its vocabulary size was 65690. The second dataset we experimented with was the Hotel Reviews Dataset, as provided to us by Lexalytics. The data contained reviews of hotels from different sources and each review was approximately 2-3 sentences in size, which accounted for approximately 36000000 tokens and more than 750000 unique words in the data's vocabulary. As can be observed, the second dataset was much larger than the first one and had a much greater key-phrase count as well. Phrases were extracted for both these datasets using methods that will be explained below. However, the government dataset was observed to have sparse phrases with not much similarity in them. This presented a challenge in grouping them together as the phrases were not concurring to common themes and too many phrases had independent themes not matching with any other phrase(s). For this reason, we discarded this dataset and carried out the remaining experiments with the reviews dataset only. A cursory overview of the dataset established that abundant similar phrases were present in it, which could be grouped under common recurring themes.

4 Preprocessing Methods

Before the raw text could be used for key-phrase extraction, certain issues had to be corrected. The Hotel Reviews dataset had several characters that violated the utf-8 encoding scheme, which was being used as the standard. Due to the presence of reviews in languages other than english, it was observed that several characters had non-standard encoding schemes, such as latin, utf-16 etc. To solve this problem, any character that caused an encoding error was replaced with an empty string. Punctuation marks were removed from the cleaned utf-8 text, along with stop words('a','an','the'), to ensure that the context being considered for the case of context-based distributed representation provided something meaningful and not standard words such as 'the' and 'a', which provide no information about the actual usage of the word/phrase.

5 Key Phrase Extraction Methods

Once we had the clean text files, the next step was to extract all the key-phrases from it. We began by extracting only the noun phrases from the text files, based on the premise that most of the times, important phrases occur in the form of noun phrases. However, due to this, we got an abundance of phrases that were not important with respect to the text, such as named entities(New York) and lost a lot of phrases(verb phrases, preposition phrases) that conveyed more information than the extracted noun phrases. For this reason, we decided to use Semantria for the purpose of key-phrase extraction. For the government dataset, we got a approximately 80000 phrases while for the Reviews dataset, we extracted close to 1700000 phrases.

Before testing any methods of phrase representation and clustering, we had to set a baseline. For that purpose, we picked a subset of 400 phrases and hand clustered those in order to prepare a human baseline to compare all our methods against. To ensure that a particular topic did not have too much bias and that each topic had more or less equal representation in the data, we limited each cluster to about 10 phrases i.e. each hand-made cluster had 10 similar phrases in it. Thus, the baseline model had 40 hand-prepared clusters based on human observation. While preparing these clusters, it was noticed that the phrases extracted from the government dataset were too sparse to cluster phrases together, which is why this dataset was neglected for all future experiments. The following table presents an example of the type of phrases that were extracted from the dataset and the manner in which they were clustered for the baseline.

Table 1: Example clusters for baseline model

Cluster 1		Cluster 2	
Short Ribs	Quick Coffee	giving workers	professional massage therapist
Hot Food	Sausage patty	excellent chef	head chef
French Toast	Sausage Rounds	talented artists	primary innkeeper
Cut fruits	dried fruits	good innkeeper	wonderful innkeepers
salad	buffet line	polite staff	accommodating staff

6 Distributed representation of Phrases

After extraction of all the phrases was complete, we had to find a method of representing the phrases that would keep their linguistic and semantic similarity intact and would allow clustering using the traditional unsupervised clustering techniques. For this, we decided to represent phrase as real valued vectors, on which vector algebra operations could be performed and clustering algorithms such as K-Means could be employed. To achieve this representation, we tried three different approaches, each of which has been explained in detail.

6.1 Phrase representation using component words

The intuition behind making use of component words to find the phrase embeddings was that every phrase can be represented in terms of the words contained in it. There are two standard ways to represent a word entity given a corpus of text - sparse one-hot encoding and dense real-valued word embedding. We chose the latter representation of words for mainly two purposes - compactness and inclusion of context. Word Embeddings using

GloVe take the make use of the word-word co-occurrence statistics to prepare word embeddings, as a result of which words which may occur in similar contexts, such as synonyms, hypernyms and hyponyms lie close to each other in the embedding space. We decided to exploit this property of word embeddings to prepare phrase representations in the following three manners:

6.1.1 Average Embeddings

In our first attempt, we took an average of the embeddings of every word present in a phrase and assigned the final vector to that phrase as its real-valued representation. For word embeddings, we used GloVe trained on a common crawl corpus, containing 1900000 words in its vocabulary. Phrases that contained at least one word not present in the GloVe vocabulary were ignored altogether. After averaging, each phrase had a 300 dimensional phrase embedding, on which the clustering algorithms chosen were applied. Results were compared with baseline to analyze if averaging word embeddings is a good way of representing phrases.

6.1.2 Concatenation

Next, instead of averaging the embeddings, we tried to concatenate embeddings of each component word together. Since each phrase can have a different number of words, to maintain uniformity, every phrase was assigned a vector of size 4*300, equal to the resultant concatenated embedding of the longest phrase in the corpus(since the size of each word embeddings was 300 and the longest phrase had 4 words in it). For phrases smaller than 4-grams, the remaining space in their embedding vector was padded with zeros. The resulting phrase embedding clusters were compared with the baseline for performance analysis.

6.1.3 Pointwise product

Finally, making use of only the component words in the phrase, we took the pointwise product of each component word embedding to prepare a real-valued representation for the phrases. This resulted in a 300 dimensional representation of each phrase, on which clustering was applied and results were analyzed.

6.2 Phrase representation using context

Drawing inspiration from skip-gram models used for preparing word embeddings, we decided to make use of the context of a phrase in learning phrase embeddings. The idea was that using context to represent phrases would prepare a distributed space where phrases occurring in similar contexts would lie close to each other. For this, we replaced, in the raw text, every phrase with a hyphenated version of it i.e. each phrase of the form "word1 word2" was replaced with "word1-word2" and plugged back into the raw text, in place of the original phrase occurrence. This was done so that in counting the vocabulary of the corpus, each hyphenated phrase would be considered as a separate unique word in itself. With this modulated text corpus, we prepared a neural network, similar to the traditional skip-gram model, for learning the embeddings of each word in the vocabulary of the corpus. Details of the network have been provided below.

6.2.1 Skip-gram Network for Word/Phrase Embeddings

A traditional skip-gram model can be seen in figure 2.

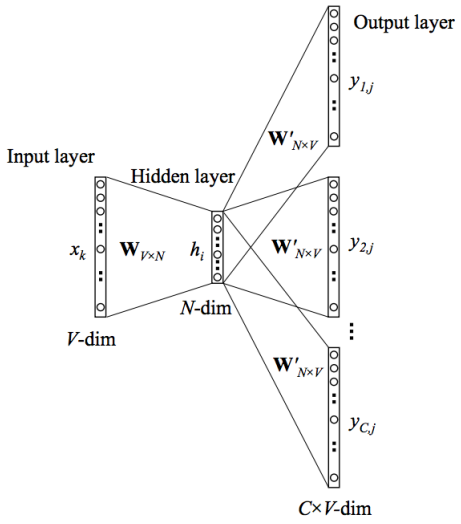

Figure 2: Skip-Gram Model as described in [2]

As you can see, the input is a one-hot encoded vector representing a single word and the output is one-hot encoded vectors of the context words occurring in the selected window for that input. Typically, the window used is of 4 words, 2 on either side of the target. The model only consists of a single hidden layer with number of neurons equal to size of embeddings desired by the user. The network learns embeddings in the following manner:

First, the weights between input and hidden layer and output and hidden layer are initialized randomly. This initialization is done uniformly between -1 and 1. Now, in each epoch(a single forward and backward pass), the network modifies its weights based on the difference between the calculated output and the actual output(one-hot vector of context words). Specifically, the cross entropy error between actual and calculated output is backpropagated using the stochastic gradient descent algorithm[1]. At the end of several iterations and multiple training examples, weights have finally been learned. The input-hidden layer weight matrix, which is a $V \times N$ matrix, where V is the vocabulary size and N is the size of embeddings required, acts as a look-up table for the N dimensional embeddings of each word present in the vocabulary. None of the intermediate layers have an activation function. However, the output layer has a softmax activation function, which can be given as follows:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for\ j = 1, ...K. \tag{1}$$

For our case, the network was trained with the available hotel-reviews corpus, with phrases replaced with hyphenated word pairs and the model was run for a total of 100000 iterations. At the end of training, embedding of each phrase was extracted and used for clustering. Traditionally, rare words are ignored or replaced with the UNK token in a skip-gram model. However, in our case, since hyphenated phrases were a rare occurrence, every word in the vocabulary was treated normally and no UNK tokens were created.

6.3 Phrase representation using both component words and context

Finally, instead of only using the component words or the context of the phrase, we decided to merge the two together and make use of both to prepare the phrase embeddings. For this, we experimented with the following two models:

6.3.1 Phrase2vec with seeded initialization of skip-gram

In the first model, we replicated the same skip-gram network prepared in the previous method. However, this time instead of randomly initializing the weight vector, the initialization was seeded with real valued embeddings for both words and phrases. For words, the weights were initialized with GloVe embeddings trained on the common-crawl corpus. For phrases(hyphenated), the weights were initialized with average embeddings, same as the ones found in section 6.1.1. The ideas behind this was that initializing with the global embeddings and learning on the local context would pull the embeddings closer to what each word/phrase represents in the present scenario, which would in turn improve the quality of information they present.

6.3.2 Feature-rich Compositional Transform

The second method we employed was inspired from [3] and can be seen in figure 3. In

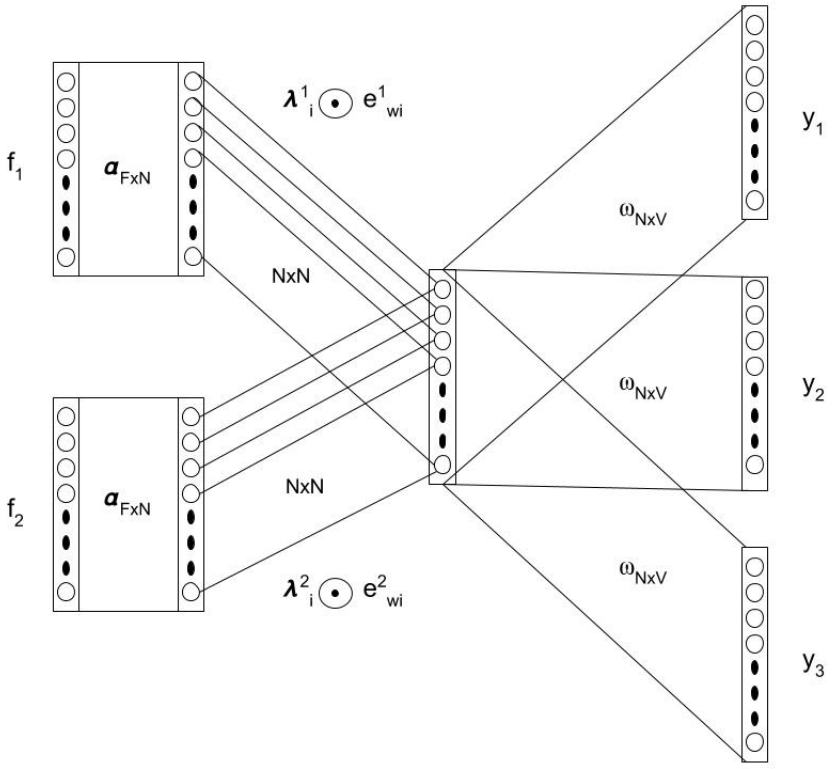

Figure 3: FCT

this method, custom features need to be prepared for every component word in a phrase. Once the features have been prepared, they are weighted and the final weighted vectors

are multiplied with that word's word embedding. To get the final phrase embedding, weighted features of all words are multiplied with their distributed representations and sum of the resultant vectors is calculated. Mathematically, this can be represented as follows:

$$e_p = \sum_{i}^{N} \lambda_i \odot e_{w_i} \tag{2}$$

Here e_p is the final phrase embedding, N is the size of the word embeddings, e_{w_i} is the word embedding of the i^{th} word and λ_i can be represented as follows:

$$\lambda_{ij} = \sum_{k} \alpha_{jk} f_k(w_i, p) + b_{ij} \tag{3}$$

where λ_{ij} presents the j^{th} value in the vector λ_i, $f_k(w_i, p)$ is the k^{th} feature of word w_i in phrase p and α_{jk} is the weight vector associated with the k^{th} feature. The symbol $\odot$ means pointwise product of two vectors.
The method explained above is called Feature-Rich Compositional Transform, due to the fact it makes use of custom features to transform phrases into their distributed representations. This method exploits information of both the words contained in the phrase and the words present in its context. In our implementation of FCT, we used the following 24 features, as shown in table 2, with only 0 and 1 as permissible values. Represented densely, these features are - POS tag of current word, POS tag of previous word, POS tag of next word, sentiment of current word, sentiment of previous word, sentiment of next word. Each word was represented using these features, which resulted in the same word having different feature vectors in different contexts. For learning the embeddings of a word, we simply took the context that occurred first and for every subsequent occurrence, used the same feature vector, regardless of its context.

Table 2: Features used in implementation of FCT

Is POS tag of word NOUN?	Is POS tag of word VERB?
Is POS tag of word ADJECTIVE?	Is POS tag of word ADVERB?
Is POS tag of word OTHER?	Is POS tag of previous word NOUN?
Is POS tag of previous word VERB?	Is POS tag of previous word ADJECTIVE?
Is POS tag of previous word ADVERB?	Is POS tag of previous word OTHER?
Is POS tag of next word NOUN?	Is POS tag of next word VERB?
Is POS tag of next word ADJECTIVE?	Is POS tag of next word ADVERB?
Is POS tag of next word OTHER?	Is sentiment of word POSITIVE?
Is sentiment of word NEGATIVE?	Is sentiment of word NEUTRAL?
Is sentiment of previous word POSITIVE?	Is sentiment of previous word NEGATIVE?
Is sentiment of previous word NEUTRAL?	Is sentiment of next word POSITIVE?
Is the sentiment of next word NEGATIVE?	Is the sentiment of next word NEUTRAL?

As in the original implementation of FCT, we seeded the word embeddings with GloVe embeddings, the same as those used in word2vec previously, and learned on these over several iterations i.e. along with learning the feature weights and biases (α and b), we also learned the word embeddings. None of the intermediate layer had any activation function, while the output layer had softmax as the activation function. After 1000000 iterations, the final values of α matrix, b matrix and embeddings were used to calculate the phrase embedding.

7 Rolling Up

Once the embeddings of phrases were prepared using the methods described above, clustering and theme roll-up were done. For clustering, as mentioned earlier, two main method were experimented with. We also attempted to roll themes up using the hypernyms and hyponyms of the phrases present in each cluster.

7.1 Clustering

7.1.1 K-Means Clustering

K-Means clustering can be defined in the following three steps:

1. In the space of points to be clustered, initialize K centroids.
2. Compute the distance of each point from each centroid and assign a point to that cluster whose centroid is closest to it. Do this for every point until preliminary clusters have been formed.
3. Within each cluster, recompute the centroid and repeat step 2 until clusters stop changing.

Here, K is the number of clusters that the user wants. The objective of K-Means is to partition N data points into K clusters in such a manner that the within-cluster sum of squares or variance is minimized. For our implementation, we made use of scikit-learn's available k-means clustering algorithm, with randomly initialized centroids.

7.1.2 Gaussian Clustering

In Gaussian Mixture Model clustering, clusters are modeled with Gaussian distributions, which means that we make use of a variance along with the mean to define each cluster. GMM allows for overlapping clusters, with the mixture model being parameterized by three values - the weight of each cluster, the mean of each cluster and the variance of each cluster. Probability of belonging to a particular cluster is assigned to each data point using an algorithm called Expectation-Maximization. Given the number of component gaussians or clusters (K), this algorithm consists of the following two steps:

1. Expectation: In the first step, the probability of each point belonging to a cluster C_k is calculated for the current values of weight ϕ_k, mean μ_k and variance σ_k of that cluster.
2. Maximization: In this step, the expectation calculated in the previous step is maximized, by modifying the values of ϕ_k, μ_k and σ_k.

This iterative model runs until convergence, at which point the maximum likelihood estimate is provided. Once the model parameters have been estimated, the fitted model can be used for clustering. A point is assigned that cluster for which the probability of it belonging to the cluster is maximum.

7.2 Cluster Labelling using Hypernyms and Hyponyms

Hypernym of a word is the superordinate of that word i.e. a word that represents the broader meaning under which other words with more specific words fall. For example, "color" is a hypernym for "red". Similarly, hyponyms are words with a more specific

meaning falling under a more general term. For example, "knife" is a hyponym for cutlery. For providing a general label to the clusters prepared using the distributed phrase representations, each phrase present in a cluster, along with its synonyms, hypernyms and hyponyms was analyzed. Each word of each phrase was checked for synonyms, hypernyms and hyponyms that were in common with other phrases. The idea was to find an intersection of ideas between multiple phrases in order to determine what the general type of phrases occurring in a particular cluster were.

In our implementation, we made use of Wordnet, a lexical database of english that provides users with information such as the synonyms etc. of a word, given the word and its part-of-speech tag. We chose to use Wordnet as opposed to any other implementation as it allows sense disambiguation i.e. it distinguishes between same words being used in different senses. Each phrase was part-of-speech tagged independently i.e. only the phrase was tagged, regardless of what its context might have been.

8 Experimentation and Results

Each method of phrase embedding extraction was tested with both clustering methods and the resulting clusters were compared with the hand-made clusters to check how well each combination of distributed representation-clustering method performed. In each method, the raw text was normalized, by lowercasing all words before any experimentation was performed. Number of clusters provided to each algorithm were in the range of [30,50] i.e. clustering was performed for each of the values in the given range. The range was decided based on the fact that when preparing hand-made clusters, it was kept in mind that each cluster will get at most 10 phrases in it, thus giving a total of 40 clusters. However, since the clustering algorithms were both automated, number of phrases per clusters were not restricted to 10. Due to this, we tested a range of 40 ± 10 when preparing clusters. The metric used for comparing manual clusters with experimental results was Cluster Purity, which can be explained as follows:
Given two sets of clusters, Purity is a measure of external evaluation that checks how pure a cluster is i.e. it checks how many points in a cluster were also present in the same cluster in the baseline model as well. Formally, we count the number of data points from the most common class in said cluster, then take the sum over all clusters and divide by the number of data points. Given, a set of cluster M and a set of classes D both partitioning N data points, purity can be presented mathematically as:

$$\frac{1}{N} \sum_{m \in M} \max_{d \in D} \mid m \cap d \mid \tag{4}$$

The purity for each case i.e. each of the 12 combinations (6 distributed phrase representation techniques and 2 clustering methods) was computed and it was found that in general, gaussian mixture models performed marginally better than K-Means clustering for all the cases. The bar chart below shows average purity of each of the 6 techniques for the case of gaussian mixture model clustering, with purity averaged over the entire range of clusters.

As can be seen, Average Embeddings performed the best, followed by word2vec with seeded initialization and FCT. Example clusters for all three have been given in tables 3, 4 and 5. We noticed that in all three best cases, the phrases seemed to be occurring together because they had common words in them, which was expected since there was not enough data for the methods to model context of the phrases properly. Another thing to notice is that methods using both component and context words performed

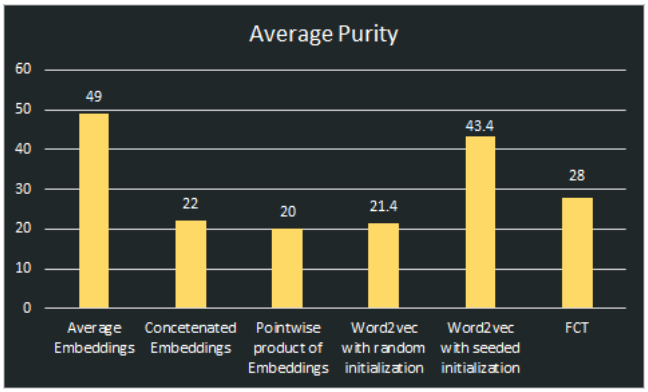

Figure 4: Average Purity for different phrase representations using Gaussian Clustering

better than the other methods that made use of only words or only context(except average), despite having limited training data.

Table 2: Examples for Average Embeddings

Cluster 1	Cluster 2
Cut fruits	living pool
dried fruits	room area
Soy Milk	parkview room
whole grains	morning coffee
fresh fruit	dining table

Table 3: Examples for Word2vec with Seeded Initialization

Cluster 1	Cluster 2
big rooms	evening drinks
Nice rooms	delicious breakfast
Family rooms	horrible breakfast - experience
modern amenities	Great meal
moldy bathrooms	lovely buffet

For labelling the resultant clusters, the hypernyms, hyponyms and synonyms of each word in each phrase in a cluster were analyzed. However, it was found that hypernyms and hyponyms did not have enough intersection among the various phrases to single out a common label for the cluster. It was also observed that more wordnet gave more relevant results when provided with a POS tag as opposed to in absence of one. However, certain words, with a particular POS tag, were unavailable in wordnet, which made it difficult to extract hyper/hypo-nyms for each word. Consequently, we abandoned this approach and focused on better representation of phrases instead.

Table 4: Examples for FCT

Cluster 1	Cluster 2
excellent breakfast	standard residence
great breakfast	queen room
great stay	night light
great meal	canine guests
terrific breakfast	standard vehicles

Conclusions and Future Work

In this project, we attempted several experiments to perform clustering of similar key-phrases together into groups. Along the process, we realized several things which we believe were key insights into the problem, that could help future researchers as well. First, we realized that there is not 'right answer' when it comes to similarity of phrases and clustering similar phrases will depend on the user as well as the cause for which clustering is being done. For example, for a user only interested in negative reviews of their hotel, clustering based on sentiment would make more sense, as opposed to the context of the phrase. Second, we realized that a good representation of phrases is almost as important as the technique of clustering used. Third, through our experiments, we concluded that for basic clustering of phrases, averaging word embeddings was a good solution, as long as the number of phrases was limited to a small value. As number of phrases to be clustered was increased, the phrases occurring in a cluster became more and more unrelated. We concluded that in our case, Average Embeddings gave a high purity because the subset of phrases we chose were distinct enough for the average embeddings to perform well. On the other hand, for a more logical and generic approach, using both context and component words of the phrase seems to perform better than only using context. Looking at the results, we concluded that with sufficient training data, phrase2vec with seeded initialization might outperform average embeddings and provide a better quality of embeddings as well. The same is true for FCT as well, which also gave the additional advantage of preparing custom features. Using these features, users may fine-tune the basis on which they want the phrases to be clustered. This would provide the user control over the kind of clusters they want for their usage.

Based on the above conclusions, there are certain steps that may be taken in the future, to improve the quality of clustering.

- More data may be acquired to train the neural network models better. With the network seeing more occurrences of the same phrase, it should prepare a better and more general phrase embedding.
- Features of FCT may be fine-tuned to improve the quality of clusters.
- Also, instead of treating each word occurrence as the same and providing the same feature vector as that of its first occurrence, average of the embeddings of all occurrences may also be taken. This means that the same word would be learned with different feature vectors and at the end of training, different embeddings each corresponding to the different phrase instances, could be averaged to get a final phrase embedding.
- For the case of labelling, extracting POS tags by including context of phrases and using these to extract hypernyms and hyponyms may provide better results as well.

References

[1] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, 1993.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[3] Mo Yu and Mark Dredze. Learning composition models for phrase embeddings. *TACL*, 3:227–242, 2015.