RT-SIFT: Concise and Meaningful Review Thread Representations

Kevin Chavez (kjchavez) Fahim Dalvi (fdalvi)

This is a joint project with CS 224N.

1 Introduction

Yelp provides valuable information for both business owners and customers, namely a view of the customers' perspectives. However, the way this information is currently presented is either too compact (such as an average star rating for the business) or too diluted—some review threads contain hundreds of reviews. Our goal is to create an intermediate representation which is concise but still reflects much of the semantic information in the review thread. This representation can then serve as a useful summary for Yelp users to quickly understand what others are saying about a business.

Yelp has a similar feature on their mobile app, titled "Review Highlights." However, it seems less sophisticated in the way it processes the actual text from the reviews. Even though it leverages additional data that Yelp has about the restaurant, such as the menu, it does a poor job of providing an overall impression of the restaurant. For example, it tells us that people are talking about "Chicken Satay", but not what they actually think about it.

2 Problem Statement

We aim to produce a representation of review threads that is concise, interpretable, and preserves much of the meaning of the full text. Further, the representation should be useful for various applications such as summarization, topic modeling, and star-rating prediction. This task can be modelled as a feature selection problem which consists of two stages: generating feature proposals and filtering/ranking these features. The first stage automatically produces a large set of human-interpretable candidate features, while the second stage reduces that set to achieve a more concise representation.

3 Related Work

There has been some work done related to analyzing reviews, primarily on summarization and opinion mining. Although many of these projects focus on specific applications, they inherently have a review representation which is used to arrive at their final results. Several research papers like [4] and [5] use Parts-of-Speech tagging and occurrence frequencies to extract features. They then "rank" or classify the features based on context and word opinion datasets, and then use these to create their summaries. Other research projects concentrate on linguistic techniques alone, like [1]. They also use Parts-of-Speech tagging, but combine it with subjectivity analysis to extract the most useful features for their summarization. Some research projects like [2] use a completely different approach, and extract their features by first selecting a set of candidate words, and finding their lexical chains. Although these projects are all focused on summarization, we can see that they all have inherent representations of the reviews (like POS tags, frequencies etc), which they then filter in order to obtain their summaries.

4 Approach

4.1 Baseline

As a baseline representation, we use a unigram bag-of-words, with Porter stemming. Each feature in this representation corresponds to a particular stemmed word. For a given review, the value of each feature is the number of times the corresponding word appears in the review. Using text from 1 million reviews to determine the feature set, we identify 69K unique stems which appear at least 5 times and appear in no more than 90% of the reviews. We evaluate the conciseness of such a representation by measuring both the maximum size and typical size of a review represented in this form. Size is defined as the number of

(word index, word count) pairs that need to be maintained. To evaluate how much meaning is preserved, we use this representation to do star-rating prediction and review thread summarization.

Representation	Max Size	Typical Size
Stemmed unigrams	69,974	53.8

Table 1: Size of baseline representation.

Rating	1	2	3	4	5
Precision	0.499	0.382	0.430	0.521	0.701
Recall	0.640	0.348	0.411	0.536	0.658
F1 score	0.561	0.364	0.420	0.528	0.679

Table 2: Baseline star rating prediction.

4.2 Oracle

Since we have two competing goals–conciseness and preservation of meaning–we present an oracle for each of them.

4.2.1 Conciseness

Suppose we only cared about a concise representation and disregarded the goal of preserving meaning. Then we can actually achieve an arbitrary level of conciseness. One way to do so is to choose a fixed size K and randomly sample K words from a review. The smallest non-trivial (i.e. not all reviews are represented identically) representation consists of a single sampled word from the review. Such a representation will lose a significant amount of information about the review.

4.2.2 Meaning

The ideal representation for this oracle would be one that encoded full information from a review, such as how a human brain might internally represent it. However, the weakness of the human brain is that it cannot hold all of this information from all million reviews "in memory." Hence, instead of a human-based system, we established that the ground truth would serve as an appropriate oracle.

These two oracles provide upper bounds on the performance of our system along each of its dimensions.

4.3 RTSift System

4.3.1 Overview

Feature generation. To represent a review, R, we propose features of the form,

$$\phi_k(\mathrm{R}) = \left|\{\text{phrase} \mid \text{phrase} \in \mathrm{R}, \text{phrase} \in \mathrm{C}_k\}\right|,$$

where C_k is the k^{th} phrase cloud (a cluster of syntactically and semantically similar phrases). In other words, each feature corresponds to a particular phrase cloud and the value of the feature is the number of times a phrase from that cloud appears in the review. The representation at this level can be simple because most of the sophistication goes into producing appropriate phrase clouds.

Feature filtering. Feature generation produces a large number of candidate features. Since we seek to produce a concise representation, it is necessary to filter and rank these features. Each proposed feature is given a score, which will later be used to determine the representation of a review. Some may be given a score of negative infinity, which indicates that they will not be part of the representation. One way to compute a score is to train a classifier that predicts the star rating of a review given these features and score models based on their contribution to classification accuracy. We tried various classifiers and ultimately decided to use a multinomial naive Bayes model because it afforded clear interpretations of the scoring metric.

4.3.2 Phrase extraction

We create phrase-level vector representations using a recursive neural network to combine word vectors according to the structure of the phrase's parse tree. Rather than using a single composition function at each node in the network, we choose the function based on the linguistic tags of the words and/or subphrases involved. Each composition function is parameterized by the weight matrix $W \in \mathbf{R}^{d \times dk}$ and the bias vector $b \in \mathbf{R}^d$, where d is the dimensionality of word and phrase vectors and k is the number of inputs to this composition (each of which is a vector in $\mathbf{R}^d$). Let g be the logistic function; then the RNN's composition functions have the form,

$$f^{(i)}(x_1, \ldots, x_k) = g\left(W^{(i)} \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} + b^{(i)}\right),$$

where i enumerates the composition functions. In the literature, this is known as a "syntactically untied" recursive neural network (SURNN). As a concrete example, consider the phrase:

the service was great

It's parse tree is shown in Figure 1. To combine the word vectors for "the" and "service" into the phrase vector for noun phrase (NP) "the service," we use a composition function $f^{(i)}$ determined by the tags DT, NN, and NP. The final RTSift system chooses a function based on the parent tag (NP for this example) and the number of children. However, we explored alternatives which produced different sized models. Ultimately, the decision to use parent tag and number of children balanced computational complexity and expressive power of our model.

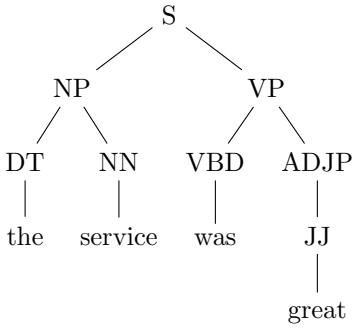


Figure 1: Parse tree for example phrase.

Training. Each training example to the SURNN is a full sentence from a review, the sentence's parse tree, and the star-rating associated with the review. We desire two properties from a fully trained SURNN. First, for any composition, we want the output phrase vector to encode information from all of its children (input vectors). This means we should be able to approximately recover the input vectors from the output vector using a simple affine transformation. Second, we wish to be able to predict the star rating associated with a sentence given the vector representations of its top level phrases. This encourages the structure of the phrase vector space to incorporate star-rating information. Thus, we define a multi-objective loss function,

$$J(T) = J_r(T) + \beta J_c(T).$$

where J_r is the reconstruction loss and J_c is the classification loss.

An important note is that many appealing properties of distributed word/phrase representations begin to deteriorate as we generate vectors for longer and longer phrases. With this in mind, we divide large trees into sub-trees such that no sub-tree has more than 6 leaves. Nodes that are above all sub-trees are ignored during training. The output of the root nodes of these sub-trees constitute the vector representations of the "top-level" phrases of a sentence.

Reconstruction loss. For every node in the parse tree, we desire that the input vectors, x_j, be well approximated by an affine transformation of the output vector, p. Specifically, we introduce new parameters $U^{(i)} \in \mathbf{R}^{dk \times d}$ and $c^{(i)} \in \mathbf{R}^{dk}$—which will also be trained—for each composition function, and define the reconstructed vectors $\hat{x}_j$ as

$$\begin{bmatrix} \hat{x}_1 \\ \vdots \\ \hat{x}_k \end{bmatrix} = U^{(i)} p + c^{(i)}.$$

We want the difference between $\hat{x}_j$ and x_j to be small. This suggests a reconstruction loss function,

$$J_r(T) = \sum_{\mathrm{n} \in T} \sum_{j=1}^{k} \left\| \hat{x}_j^{(n)} - x_j^{(n)} \right\|_2^2,$$

where the summation variable n represents a node in the parse tree T.

Classification loss. The second desired property is the ability to predict the star-rating corresponding to a *sentence* given the vector representations of the top-level phrases. We augment the SURNN with a softmax layer on top of the root nodes and introduce the parameter

$$\theta = \begin{bmatrix} \theta_1^T \\ \vdots \\ \theta_5^T \end{bmatrix} \in \mathbf{R}^{5 \times d}.$$

Given the vector representation at m root nodes, $r_1, \ldots, r_m$, this layer predicts a distribution over star-ratings,

$$p(s \mid r_1, \ldots, r_m) = \frac{e^{\theta_s^T (r_1 + \cdots + r_m)}}{\sum_{\ell=1}^{5} e^{\theta_\ell^T (r_1 + \cdots + r_m)}}.$$

Note that there is an implicit linearity assumption about the space of top-level phrases within a sentence. In particular, we assume that the sum of

phrase representations roughly represents the presence of all of those phrases. Using the distribution above, we define the classification loss J_c as a regularized version of the logistic loss function,

$$J_c(T) = -\sum_{s=1}^{5} \mathbf{1}\{s = S\} \log p(s \mid r_1, \ldots, r_m) + \lambda \|\theta\|_{\mathrm{F}}^2,$$

with the regularization parameter λ.

Local losses and gradients. We use stochastic gradient descent to minimize the loss function over our training set. To do so, we must compute the gradient of $J(T)$ with respect to the parameters $W^{(i)}$, $U^{(i)}$, $b^{(i)}$, $c^{(i)}$, and θ. Note that these gradients depend heavily on the structure of the parse tree, so let's define a few quantities that are local to each node in the tree. Consider a node n with k children which uses the composition function $f^{(i)}$. Let $J_r^{(n)}$ be the *local reconstruction loss*,

$$J_r^{(n)} = \sum_{j=1}^{k} \left\| \hat{x}_j^{(n)} - x_j^{(n)} \right\|_2^2.$$

For notational convenience let g be the output vector of this node, $g' = g \circ (\mathbf{1} - g)$ be the derivative of the logistic function with respect to its input, and $x^{(n)}$ be the vertical concatenation of all children vectors. Finally, let r be the residual $r = U^{(i)} g + c^{(i)} - x^{(n)}$. The gradients of the local loss function with respect to the parameters used by this node are

$$\nabla_{W^{(i)}} J_r^{(n)} = 2(U^{(i)T} r \circ g') x^{(n)T} \qquad \nabla_{U^{(i)}} J_r^{(n)} = 2 r g^T$$
$$\nabla_{b^{(i)}} J_r^{(n)} = 2 U^{(i)T} r \circ p' \qquad \nabla_{c^{(i)}} J_r^{(n)} = 2r$$

where the symbol $\circ$ denotes the Hadamard, or element-wise product. For backpropagation, we will also need the gradient of the local reconstruction loss with respect to the inputs to this node,

$$\nabla_{x^{(n)}} J_r^{(n)} = -2r + 2 \left(U(g' \mathbf{1}^T \circ W^{(i)}) \right)^T r$$

For the nodes that are roots of sub-trees, we can also compute simple local gradients of the classification loss with respect to the output vector of the root node,

$$\nabla_{r_i} J_c = \theta^T (y - \hat{y})$$

where y is the delta distribution on the correct star rating of the sentence (for example, if it had a rating of 2 then $y = [0; 1; 0; 0; 0]$) and $\hat{y}$ is the distribution predicted by the softmax model $p(s \mid r_i)$.

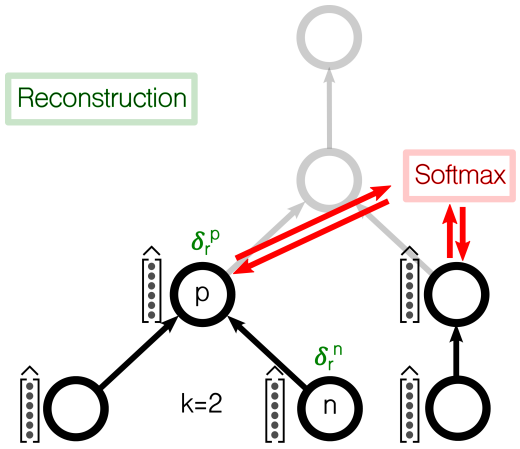


Figure 2

It's important to note that the local reconstruction loss at a node is affected by parameters used by descendant nodes, but it is not affected by what happens in its ancestors. Therefore we must propagate each local reconstruction error only through it's subtree.

Backpropagation. We will only consider how to propagate the reconstruction loss. The treatment of classification loss is more straightforward since the loss function is only defined on "output nodes" (nodes that don't feed into any others). See, for example, [3].

Let's define a variable $\delta_r^{(n)}$ for every node n as

$$\delta_r^{(n)} = \nabla_{x^{(n)}} J_r$$

where $x^{(n)}$ is again the vertical concatenation of all the input vectors to node n. If we know the value of $\delta_r^{(p)}$ for this node's parent, p, and we know that n is the k^{th} child of p, then we can efficiently compute $\delta_r^{(n)}$ as the sum of the local gradient and a term that depends on $\delta_r^{(p)}$,

$$\delta_r^{(n)} = \nabla_{x^{(n)}} J_r^{(n)} + \left(g' \mathbf{1}^T \circ W^{(i)} \right)^T S_k \delta_r^{(p)}, \qquad (1)$$

where S_k is a selection matrix which effectively selects the $((k-1)d + 1)^{\text{th}}$ to kd^{th} elements of $\delta_r^{(p)}$. (Reminder: d is the dimensionality of word/phrase vectors.) The first term is simply the local gradient that we have described in the previous section. To see where the second term comes from, it's best to use a different notation for the gradients. In particular,

$$\nabla_{x^{(n)}} J_r = \frac{\partial J_r}{\partial x^{(n)}} = \frac{\partial J_r^{(n)}}{\partial x^{(n)}} + \frac{\partial \sum_{m \neq n} J_r^{(m)}}{\partial x^{(n)}}$$

Let's rewrite the second term as

$$\frac{\partial \sum_{m \neq n} J_r^{(m)}}{\partial x^{(n)}} = \left(\frac{\partial x^{(p)}}{\partial x^{(n)}}\right)^T \frac{\partial \sum_{m \neq n} J_r^{(m)}}{\partial x^{(p)}}.$$

Note that since p is the parent of n, the local reconstruction loss $J_r^{(n)}$ doesn't depend on $x^{(p)}$. Therefore

$$\frac{\partial \sum_{m \neq n} J_r^{(m)}}{\partial x^{(p)}} = \frac{\partial \sum_{m \neq n} J_r^{(m)}}{\partial x^{(p)}} + \frac{\partial J_r^{(n)}}{\partial x^{(p)}} = \frac{\partial J_r}{\partial x^{(p)}} = \delta_r^{(p)}.$$

It can be shown that

$$\left(\frac{\partial x^{(p)}}{\partial x^{(n)}}\right)^T = \left(g' \mathbf{1}^T \circ W^{(i)}\right)^T S_k,$$

which brings us to the form in (1). For any output node, t, that doesn't feed into any others, $\delta_r^{(t)}$ only consists of the gradient of the local reconstruction loss, with respect to $x^{(t)}$

Global gradients. Let $(\nabla_\mu J_r)_n$ be the contribution to $\nabla_\mu J_r$ from node n. Note J_r is the total reconstruction loss, not just the local loss. In other words

$$\nabla_\mu J_r = \sum_{n \in \text{tree}} (\nabla_\mu J_r)_n,$$

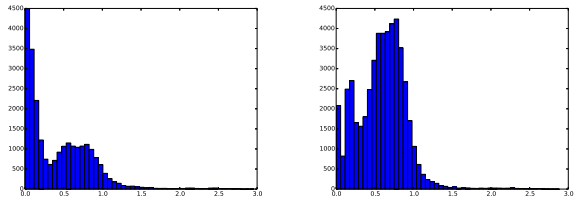
for any parameter μ. Once we have computed $\delta_r^{(n)}$ for each node, we can also compute the node's contributions to the gradients of J_r with respect to the parameters of our model. We state these here without the derivation. Suppose node n used the composition function $f^{(i)}$. Then

$$\begin{aligned}
(\nabla_{W^{(i)}} J_r)_n &= \left(S_k \delta_r^{(p)} \circ g'\right) x^{(n)T} + \nabla_{W^{(i)}} J_r^{(n)} \\
(\nabla_{b^{(i)}} J_r)_n &= \left(S_k \delta_r^{(p)} \circ g'\right) + \nabla_{b^{(i)}} J_r^{(n)} \\
(\nabla_{U^{(i)}} J_r)_n &= \nabla_{U^{(i)}} J_r^{(n)} \\
(\nabla_{c^{(i)}} J_r)_n &= \nabla_{c^{(i)}} J_r^{(n)}.
\end{aligned}$$

Once the SURNN is trained, we can compute vector representations for all phrases in our dataset.

4.3.3 Phrase cloud formation

We extract phrases of lengths 2 through 6 from the reviews for all 42K businesses. Using a subset of 2.5 million of these phrases, we use K-means to partition the phrases into 50,000 clusters. The motivation for choosing a particular value of K comes from the corresponding cluster radii distributions. Figure 3 shows radii distributions for two values of K. Notice that for $K = 25,000$, the radii distribution is bimodal, whereas for $K = 50,000$, we have a unimodal distribution. A unimodal distribution indicates consistently sized clusters, which partitions the space in an unbiased manner.



25K clusters 50K clusters

Figure 3: Comparison of radii distributions.

For each cluster (also referred to as a phrase cloud), we store the centroid and the distance to farthest member of that cluster seen during training. This defines the volume of the phrase vector space that corresponds to that cloud.

4.3.4 Phrase cloud scoring

To score the proposed phrase cloud features, we use an entropy-based value. Across our set of review data, we have around 57 million phrases, each with an associated star-rating from its parent review. We assume that given a particular star rating for a review, the presence of any of these phrase clouds is independent of any other. Thus, we can compute the probabilities $p(c_i \mid \text{rating} = s)$ for each phrase cloud c_i (this is equivalent to training a Naive Bayes model to predict star ratings over our data). Using Bayes rule and the prior distribution $p(\text{rating} = s)$, we can compute the conditional probability of a review having a particular star rating given the presence of a phrase from phrase cloud c_i,

$$p(\text{rating} = s \mid c_i) = \frac{p(c_i \mid \text{rating} = s) p(\text{rating} = s)}{p(c_i)}$$

We use the entropy of this distribution to measure the relevance of a phrase cloud. Note that here we are leveraging labels for star rating prediction, but the phrase clouds themselves have more general structure in them due to the reconstruction objective imposed during phrase extraction. Thus, the score reflects how well a group of structurally and semantically similar phrases align with a particular type of experience as indicated by the star rating. To be

concrete, the score of a phrase cloud is defined as the negative entropy

$$\text{Score}(c_i) = \sum_{s=1}^{5} p(\text{rating} = s \mid c_i) \log p(\text{rating} = s \mid c_i).$$

Greater scores indicate distributions that are more focused on a single star-rating. Low scores indicate diffuse distributions which suggests mal-formed or less relevant phrase clouds.

For some phrase clouds, the score might be high due to insufficient data. In the extreme case, suppose only one phrase from the 57 million belonged to this phrase cloud and it happened to come from a review with star rating 4. Then our distribution over star ratings would be well-peaked at 4 and would receive a high score, even though such a phrase cloud empirically only has a $1.75 \times 10^{-6}\%$ chance of occuring. Ideally, with well-formed phrase clouds this shouldn't be a problem. However, we have seen cases while testing that suggest otherwise. Thus we place a threshold on $p(c_i)$. If a cloud is below this threshold, its score is set to $-\infty$.

4.3.5 Representing a review

Let k be the number of phrase clouds with a score greater than negative infinity. We represent a review consisting of the set of phrases P as a vector $\phi(R) \in \mathbf{R}^k$ of counts of phrases from each the k phrase clouds. Precisely, similar to the form described in section 4.3.1,

$$\phi_k(\text{R}) = \Big|\{\text{phrase} \mid \text{phrase} \in \text{R}, \text{phrase} \in C_k\}\Big|.$$

The difference is that we have filtered out a fair number of phrase clouds.

5 Data

Out primary dataset was from the *Yelp Academic Dataset Challenge*. It had around $1,125,458$ reviews for approximately $42,000$ businesses. For each of the review, we had the full text for the review (which we then tokenized into $10,194,645$ sentences) and the star-rating associated with each review. For the word vectors, which were the basis for our phrase vectors, we decided to use 50-dimensional word vectors from the *Global Vectors for Word Representation* (GloVe) project.

6 Experiments and Results

Throughout the project, we encountered several challenges and findings that forced us to rethink our design choices.

6.1 SURNN

6.1.1 Loss function

The loss function presented in Section 4.3.2 is one of several variants tested. The alternative loss functions were "reconstruction loss only" and "reconstruction and phrase-level classification loss."

Reconstruction loss only. Our original formulation only involved a reconstruction loss function at every node of the parse tree. The goal was that the vector representation of the children was approximately linearly decodable from that of the parent. Intuitively, the word vectors encode some semantic information and the reconstruction objective tries to reduce the loss of information as multiple words are combined. However, after creating phrase clouds based on this system, we see too many instances where the cluster of phrases have similar syntactic structure, but express wildly different sentiments. For example, "a burnt crust" and "a golden crisp crust" are actually quite similar, but the latter is a much more positive experience than the former. Phrase clouds that do not have a focused star-rating orientation are not very helpful as part of our final representation. Thus, we sought to weakly propagate (see 6.2 for various levels of propagation) the star-rating labels into phrase vector space to encourage better-formed phrase clouds.

Phrase-level classification. During training, large parse trees are divided into manageable subtrees. Thus, for each sentence we have a set of root nodes corresponding to top-level phrases. One way to include star-rating information in the loss function is to place a softmax layer on the root of each sub-tree, which tries to predict the star-rating for the entire review. This implicitly assumes that the star-rating of a review should be assigned to *each* of its constituent phrases.

6.2 Phrase clouds

While creating the phrase clouds, we had several design choices to make. One example was how strongly

did we want to propagate the star-rating information into the phrase vector space. From 4.3.2, we see that the β parameter helps us control exactly this. We tried various values for β, and finally decided to use $\beta = 1$. We also tried various number of phrase clouds, finally using $50,000$ clouds (motivated by the cluster radii distribution). Because we were computationally bound, we reduced the number of phrases used for clustering to 2.5 million. We had to compromise on how which phrases to include. One option was to use only phrases with VP (verb phrase) or FRAG (sentence fragment) tag. These seemed to typically contain more useful information, and this way we could include phrases from around $11,000$ businesses (out of $42,000$). However, in later testing, we realized that representations that only included VP and FRAG phrases missed critical information. Thus, we expanded to using phrases with all tags, even though it meant we could only process around $2,000$ businesses. We later modified our clustering to work out-of-memory, but did not have time to train a model on a significantly larger batch of phrases.

Analyzing the phrase clouds visually (we used *Principle Component Analysis* to reduce the dimensionality), we noticed the existence of clusters:

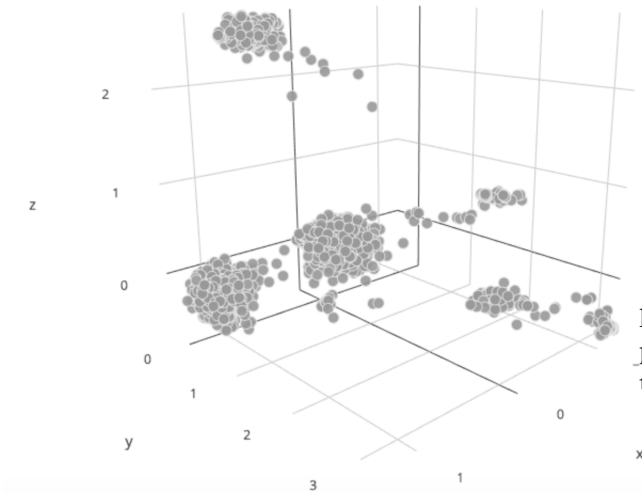


Figure 4: Twenty thousand phrase vectors projected into 3D show non-trivial structure.

For these clusters to be meaningful, we would like similar clouds of phrases to be closer together in the vector space, and those that are most dissimilar to be far apart. If this structure exists, we can build a useful distance metric over review representations. While we do not cluster the phrase cloud centroids explicitly, we analyze the distribution of pairwise distances between cluster centroids. The histogram in Figure 5 shows that the distribution of distances is bimodal. This distribution implies the existence of super-clusters of our phrase clouds.

As mentioned earlier, we tried Softmax Regression, Naive Bayes and a Softmax layer over the SURNN to rank phrases/phrase clouds by their importances. One idea would be to combine these methods together to form an aggregate rank, which may better serve to identify important phrase clouds. It might also be fruitful to try different clustering algorithms other than K-Means, as we notice some bad phrase clouds even with 50,000 clusters.

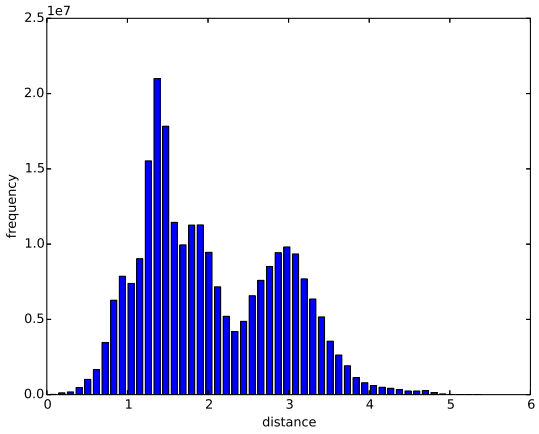


Figure 5: Distribution of pairwise centroid distances

Analyzing some of the phrase clouds, we notice cohesiveness in syntax as well as semantics among the phrases. An example of a high scoring five-star cluster contains:

> *a stellar job, a fantastic job, an incredible job, a super job, a lasting job, an amazing job*

6.3 Representation size

Since one of our goals was to achieve a concise representation, we measure the average size of a representation created by the RTSift system. Over a sample of $15,300$ businesses (approximately 36% of the full dataset), the average size of a representation is 6.11 (phrase index, phrase count) pairs. Compared

to the typical size of our baseline representation, this is almost a 9-fold reduction. The RTSift system has a memory overhead due to maintaining the cluster centers, but this cost does not scale with the number of reviews represented.

6.4 Star rating prediction

With only a small subset of the data used to create phrase clouds (only 2.5 million out of 57 million phrases or around 4.5%), we show that star rating prediction scores are competitive with the baseline model, which generated features by using the entire dataset. The confusion matrix for the RTSift model is shown in Figure 6; precision, recall and F1 scores are in Table 3.

Rating	1 star	2 star	3 star	4 star	5 star
Precision	0.527	0.399	0.439	0.520	0.680
Recall	0.556	0.339	0.417	0.554	0.669
F1	0.541	0.366	0.428	0.536	0.675

Table 3: Classification scores for $\beta = 1.0$ RTSift model.

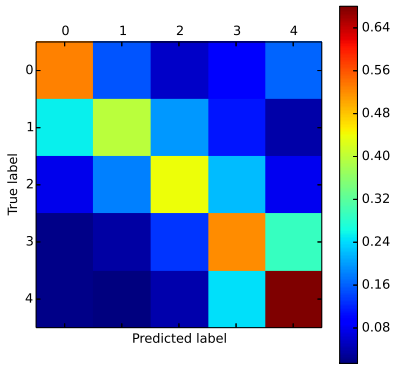


Figure 6: Confusion matrix for star rating prediction

Even though our scores are not very high, we can see from the confusion matrix that the majority of the errors are near to the diagonal. Hence, the predictions are close to the true predictions, and can be used as a reliable metric for ranking/filtering our features.

6.5 Summarization

We experimented with different sizes for the summaries that we generated, and qualitatively established that 5 sentences is an appropriate length of what one expects from a summary for review threads. For getting quantitative results for summarization as an application of our system, we conducted a survey. In the survey, we choose relatively short review threads (20-30 sentences) which each had between 3 to 5 reviews. We then present the users with three summaries (each 5 sentences long), and ask them to choose which of the three choices summarizes the review thread best. The three choices were Random, First-Last and RTSift. Random was just picking random sentences from the review thread. First-Last was a random sample from the set of all first and last sentences from every review in the review thread. Finally, RTSift was the summary from our system. We received a total of 36 responses, with the following results:

System	#Responses	%
Random	12	33.33%
FirstLast	13	36.11%
RTSift	11	30.56%

Table 4: Summarization survey results.

Although the results do not look promising, there are a few things to keep in mind:

1. The model used for the survey only used VP and FRAG as potential features, in order to reduce our processing time.

2. The reconstruction-classification trade-off parameter β has not been very finely tuned, again because of time constraints. Trying different values of β requires us to re-run the entire pipeline, which runs in the order of a day.

3. The K-means model was also trained on only 15% of the set of all VPs and FRAGs, equivalent to only 4.5% of all phrases. Even though we optimized the K-means phase by using Mini-Batch K-means, the runtime was still a limiting factor.

Therefore, we believe that resolving these issues (either by optimizing or using more time) should give us better summaries.

7 Analysis

7.1 Training SURNN

We analyze how the average sentence loss varies with the amount of training data primarily to judge the efficacy of continued training. The average sentence loss consists of both the total reconstruction loss across all nodes and the classification loss for the sentence. Consistently, it seems that our loss on the development set hits a plateau after training on around 4000 review threads (equivalent to roughly 75,000 reviews or 9 million words).

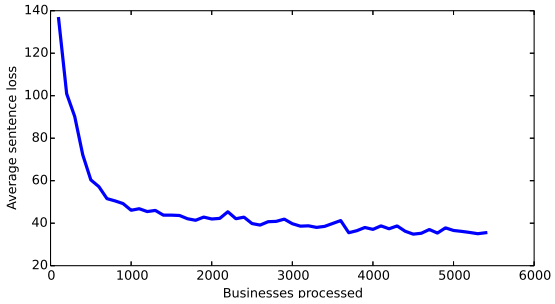


Figure 7: Training on larger and larger datasets.

We considered two major possible sources for this behavior. First, the stochastic gradient descent algorithm could be stuck in a poor local minimum or in a region where the gradients are all very small, causing progress to be slow. Restarting training with different random initializations of the parameters show that in fact, the minimum that the system in Figure 7 is approaching actually better than other minima (though we do not know how far from the global minimum this may be).

7.2 System errors in applications

7.2.1 Star-rating prediction

When using the RTSift representation, the naive Bayes classifier does worst on 2-star reviews, achieving a precision and recall of 0.399 and 0.339. From the confusion matrix in section 6.4, we can see that the system most frequently mistakes a 2-star review for a 1-star review. To dive deeper into the source of the error, consider the following representative example:

> we called ahead to make sure they were able to dry clean our two shirts on time before the following afternoon. they were fast and friendly during drop-off and pick-up, but when my boyfriend put on the shirt after flying out of town the next day we realized it hadn't been cleaned at all. there were still underarm stains and full body odor on the shirt despite the shirt being pressed nicely. perhaps they didn't have time and didn't care to tell us because we mentioned we had a flight to catch the next day so they assumed we weren't locals. it was a bizarre experience and we will not be using their service again.

Some of the phrases the system deemed most important in this review are: "using their service again" "cleaned at all" and "was a bizarre experience." However, even these clusters have a relatively low score as defined in Section 4.3.4. This low score implies that the corresponding phrase cloud is not well-formed. For example, if we look at the phrase cloud for "was a bizzare experience," we see phrases such as "was a horrible experience" and "was a thoroughly satisfying affair." Although all three have similar meanings in some sense, they are widely different with respect to implied star-rating. In many cases of poor performance, this is a major cause.

7.2.2 Summarization

Occasionally, reviewers will include detailed stories about their visit to the restaurant or about their personal history (see the third review in "data/processed/threads/b00155" for an example). These sections of text are not usually useful for representing a review or review thread. However, because they are typically long stories, they will contribute phrases to the representation and degrade its quality. Often this results in poor summaries, but more importantly poor representations. Since we treat every sentence in a review identically when training our representation system, we cannot address this source of error. One way to make progress is to include a layer of subjectivity analysis on the sentence level which can help identify which sentences we should be drawing our representation from.

8 Future Work

Our goal throughout the project was to create a review representation that is generalizable, so that it can be used in many different applications. Apart from the summarization that was implemented as part of the project, some possible future applications include:

Business Comparison. The review representation can be used to compare the key similarities and differences between any given businesses, and can thus help suggest changes a business should make in order to improve its average star rating. Since the representation is based on reviews, this procedure would help capture hidden insights that customers talk about often, not just superficial differences (often found in meta-data) that may or may not be extremely useful.

Recommendation Systems. We could also use the review representations of past choices that a user has made, and compare it against future choices to predict what business the user would enjoy the most. Again, this is not done with already available data like business type and keywords, but rather with insights from customer reviews.

Topic modeling. Similar to the summarization application, we could also perform topic modeling on a review thread to grasp key ideas related to a business.

Other than the applications themselves, there are key ideas that might potentially improve the final representation itself. For example, as mentioned earlier, we tried Softmax Regression, Naive Bayes and a Softmax layer over the SURNN to rank phrases/phrase clouds by their importances. One idea would be to combine these methods together to form an aggregate rank, which may better serve to identify important phrase clouds. It might also be fruitful to try different clustering algorithms other than K-Means, as we notice some bad phrase clouds even with 50,000 clusters.

On the data side of things, it would be ideal to train on more data, especially K-Means so that it can form better phrase clouds. Experimenting with more or less clusters would also be helpful to find overall better clouds. Also, currently to keep the processing time reasonable, we cutoff the parse trees both while creating the SURNN structure and while extracting phrases in the pipeline. We have not experimented with various levels of cutoff. We also consider only specific tags like NP or VP. The ideal situation would be to consider the complete tree with all the labels and no cutoff.

Finally we can also improve the application considered in this project, summarization. An improvement can be expected by performing subjectivity analysis on a sentence level, so we choose better candidates for our summary. Taking care of dangling anaphoric references would also improve the quality of the summary, as currently sentences picked from anecdotes in the reviews usually have dangling anaphoric references.

References

[1] Abulaish, M., Jahiruddin, Doja, M., and Ahmad, T. (2009). Feature and opinion mining for customer review summarization. In Chaudhury, S., Mitra, S., Murthy, C., Sastry, P., and Pal, S., editors, *Pattern Recognition and Machine Intelligence*, volume 5909 of *Lecture Notes in Computer Science*, pages 219–224. Springer Berlin Heidelberg.

[2] Barzilay, R. and Elhadad, M. (1999). Using lexical chains for text summarization. *Advances in automatic text summarization*, pages 111–121.

[3] Elkan, C. (2014). Learning Meaning for Sentences.

[4] Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 168–177, New York, NY, USA. ACM.

[5] Liu, B., Hu, M., and Cheng, J. (2005). Opinion observer: Analyzing and comparing opinions on the web. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 342–351, New York, NY, USA. ACM.

[6] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

[7] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12.

[8] Socher, R., Huang, E. H., Pennin, J., Manning, C. D., and Ng, A. Y. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.