An Entropy-Based Comment Ranking Method with Word Embedding Clustering

Yuyang Zhang and Hao Yu

Abstract Automatically ranking comments by their relevance plays an important role in text mining. In this chapter, we introduce a new text digitization method: the bag-of-word clusters model, i.e., grouping semantic-related words as clusters using pre-trained word2vec word embeddings and representing each comment as a distribution of word clusters. This method extracts both semantic and statistical information from texts. Next, we propose an unsupervised ranking algorithm that identifies relevant comments by their distance to the "ideal" comment. This "ideal" comment is the maximum general entropy comment with respect to the global word cluster distribution. The intuition is that the "ideal" comment highlights aspects of a product that many other comments frequently mention. Therefore, it is regarded as a standard to judge a comment's relevance to this product. At last, we analyze our algorithm's performance on a real Amazon product.

Keywords Bag-of-word clusters · Comment ranking · Cosine similarity · General entropy · K-L divergence · Word2vec

1 Introduction

Online shopping has become popular all over the world. The most obvious benefit of online shopping is convenience, and shoppers can simply access online stores from their computers whenever they have free time available, in particularly, during current pandemic time. Another benefit is that online shopping provides a greater diversity of products. This means you can choose goods that suit your requirements

Y. Zhang · H. Yu (✉)
Department of Statistical and Actuarial Sciences, University of Western Ontario, London, ON, Canada
e-mail: yyzhang9603@gmail.com; hyu@uwo.ca

W. He et al. (eds.), *Advances and Innovations in Statistics and Data Science*, ICSA Book Series in Statistics, https://doi.org/10.1007/978-3-031-08329-7_5

and budget the most. However, there are also disadvantages of online shopping. One of the most obvious ones is the lack of interactivity. You cannot touch and feel the product you want to buy. Besides, the lack of touch and feel creates concerns over the quality of the product. With a large variety of goods and websites, people tend to do a lot of research before making a purchasing decision when doing online shopping. They will browse web pages about product details and, more importantly, check other buyer's comments on the product site.

Gathering information based on other people's opinions is an essential part of the purchasing decision process (Chevalier & Mayzlin 2006). With the rapid growth of the Internet, these conversations in online markets provide a large amount of product information. So when doing online shopping, consumers rely on online product comments, posted by other consumers, for their purchase decisions.

However, a large number of comments for a single product may make it harder for people to evaluate the true underlying quality of a product. In this situation, consumers tend to focus on the average rating of a product, like the number of stars on Amazon.com. But in reality, some products can easily obtain high average ratings by cheating, while some other products may get unfair low ratings. Therefore, it is very important to extract these relevant and high-quality comments from the product site, which help consumers obtain accurate information about this product.

1.1 How to Judge a Comment's Quality?

Before we start to construct a comment ranking algorithm, the fundamental question is how to judge a comment's quality. Most online business sites evaluate their comments' quality using criteria such as overall rating or helpfulness. Helpfulness is typically a score measured as the total votes given by consumers, which is an interesting way of defining a comment's relevance and quality. Many researches in comments ranking area also use this type of helpfulness score as their comments' evaluation score (Zhang & Varadarajan 2006). However, this method fails to identify these most recent comments with few votes. For example, we may always observe that only a few comments published a long time ago have a high helpfulness score in a product site, and most other comments have no votes. The reason for the phenomenon is that most people only read the first few pages of comments before making their purchase decisions. A new comment that has just appeared on the product site and has not received any votes until recently may remain at the bottom of the comment list. This comment may contain important information about this product and thus has the potential to rise to the top of the list.

1.2 Ranking Comments Using Entropy

Ranking comments is a very important task, and there is no doubt that there are many studies in this field. Some researches treat this ranking task as a supervised

learning task, such as Hsu et al. (2009) and Alberto et al. (2015). Most of them used the consumers' votes, such as helpfulness scores, as their training target. Then they adopted or designed several statistical or machine-learning models based on training data. As mentioned before, the reliability of this training data is hard to be assured; some high-quality comments may have relatively low votes. Moreover, these supervised ranking models cannot be used on multiple products simultaneously since they have to be retrained for different products.

In this situation, we propose a comment ranking algorithm that is **unsupervised**, which means we do not require any human-annotated training set. Besides, as we mentioned before, in most cases, online business sites may not be able to provide some information, such as the reviewer's reputation. We prefer to develop an algorithm that ranks comments based on their contents. To construct this ranking algorithm, we need to solve these two problems below.

1. How to define a metric that can evaluate both comments' relevance and text complexity?

2. How to effectively retrieve information from comments' content?

Let us take a look at the first question first. When it comes to text complexity or information richness, we naturally think of **Shannon's entropy** (Shannon 1948). However, **Shannon's entropy** does not measure comment's relevance to the product. So how to redefine entropy and take comment's relevance into account? Zhang (2019) defined a new entropy value called **the general entropy**. In his thesis, he developed an unsupervised ranking method on Amazon's dataset and used the general entropy to measure the answer's information quality. The general entropy is defined as follows:

$$E(\mathbf{P}) = -\sum_{i=0}^{n} Q_i \cdot P_i \cdot \log P_i, \tag{1}$$

where $\mathbf{P} = [P_0, ..., P_n]$ is the words' distribution of an individual comment and $\mathbf{Q} = [Q_0, ..., Q_n]$ is the distribution of the words of all comments combined under the same product, which is called global distribution. So general entropy assigns weight on each self-information of word where the weight is the corresponding word probability in the global comment set. Since we want to measure the information richness and the relevance of a comment, we give higher weight to these words that other comments also mentioned and lower weight to words that other comments hardly mentioned.

The general entropy seems a good ranking metric that measures both relevance and text complexity. However, comment with high complexity (e.g., very long comment with many different kinds of words) and almost no relevance to this product may get very high general entropy. These comments may have high ranks since most comments' entropy scores are close to each other. So instead of calculating scores for every comment, we first find an "ideal" comment and then judge comment by how far or how different it is from our "ideal" comment. Naturally, we define a comment with maximum general entropy as our "ideal" comment, and we call this "ideal" comment the **Maximum General Entropy**

Comment. Since this "ideal" comment has the maximum general entropy, it keeps a good balance of relevance and information richness. We define the **Maximum General Entropy Comment** as follows:

$$\mathbf{B} = \arg\max_{\mathbf{P}} E(\mathbf{P}). \tag{2}$$

Note that the maximum entropy comment is a comment with the maximum general entropy within all possible comments. This comment may not exist in the existing comment set. Now, the question is how to measure each comment's "distance" to this "ideal" comment. Since we treat each comment as a distribution, we use **Kullback–Leibler (K-L) divergence** defined as follows:

$$D_{KL}(\mathbf{P}|\mathbf{B}) = \sum_{i=0}^{n} P_i \cdot \log\left(\frac{P_i}{B_i}\right). \tag{3}$$

Notice that this is a divergence, not a "distance." Actually, K-L divergence is used to measure how one probability distribution is different from others. In our application, we need to compare how each comment is different from the maximum general entropy comment. If a comment is similar to the maximum general entropy comment, it gets low divergence. Otherwise, if a comment is very different from the maximum general entropy comment, it may get high divergence. Compared to the method purely using the general entropy, this method achieved better performance in our experiment since it is more sensitive to comments' relevance to the product.

1.3 Text Representation

Let us consider the second problem described before: **How to retrieve information from comments' content effectively?** As we discussed in the previous subsection, in order to use entropy as our comments' ranking metric, we need to treat each comment as a distribution of words $\mathbf{P} = [P_1, ..., P_n]$. $\mathbf{P}$ is a numerical vector where each dimension indicates the frequency of a word that appeared in the comment, and n indicates the number of unique words in the whole collection of comments. This is actually called the bag-of-words (BOW) model. A bag-of-words is a representation of text that describes the occurrence of words within a document. Despite the simplicity of this representation method, it has two significant disadvantages:

(1) The number of unique words in comments data is about 10,000, while each comment has only 10–200 words, and using the BOW model leads to high-dimensional and sparse vectors.
(2) BOW representation does not consider the semantic relation between words, and it assumes all the words are independent. This assumption may have some problems, for example, "used bicycle" and "old bike" will be considered entirely different phrases because they have no words in common.

In this chapter, we construct an entirely different method to solve both the BOW model's sparsity and the semantic problem. We called our model: the bag-of-word clusters model. Unlike the traditional bag-of-words model that treats each word as an independent item, we group semantic-related words as clusters using pre-trained word2vec word embeddings (see Mikolov et al. 2013a,b). For example, consider the "used bike" and "old bicycle" example, and we have four unique words: "old," "bike," "used," and "bicycle." By using the traditional BOW model, we represent "old bike" as vector $[\frac{1}{2}, \frac{1}{2}, 0, 0]$ and represent "used bicycle" as $[0, 0, \frac{1}{2}, \frac{1}{2}]$. It turns out that these two vectors are orthogonal, and two vectors have no elements in common. But in reality "used bicycle" and "old bike" are semantic-related. Using our methods, we first group similar words such as "bicycle" and "bike" into the same cluster and treat them as the same item. For example, we have two groups: cluster #1: "used," "old"; cluster #2: "bike," "bicycle." Then we represent "old bike" as $[\frac{1}{2}, \frac{1}{2}]$, where each dimension indicates one cluster, and then the bag-of-word clusters representation of "used bicycle" is also $[\frac{1}{2}, \frac{1}{2}]$. Using this example, our method solves the BOW model's sparsity problem, and the number of clusters is significantly smaller than the number of unique words. Also, we retrieve semantic information from text, and similar phrases "old bike" and "used bicycle" are represented as the same vector in our model. Besides, unlike Zhang's method, which only considers keywords and treats all other words as noise, our method keeps most words and treats related words as the same item. We believe our method can extract more information from text and thus has a better ranking performance. More detail about the bag-of-word clusters model is introduced in the next section.

The rest of this chapter is organized as follows. In Sect. 2, we propose a new text representation method: the bag-of-word clusters model. In Sect. 3, we give a detailed description of our ranking algorithm. Section 4 introduces our experiment with a real Amazon product using the Amazon product dataset (see He and McAuley 2016; McAuley et al. 2015).

2 Bag-of-Words Model with Word Embedding Clusters

In most text mining or text analytics applications, the first and fundamental problem is how we represent text as input to our model or algorithm. More specifically, how do we represent the text documents to make them mathematically computable? Various text representation methods were proposed during the last few years, and the most commonly used text representation model in the area of text mining is called the vector space model (VSM) (Manning et al. 2008), which aims to represent a text document as numerical vectors. One main advantage of VSM is that it is straightforward to compute the similarity between each vector (document), for example, by using cosine similarity.

One of the commonly used VSMs is the BOW. A bag-of-words is a representation of text that describes the occurrence of words within a document. And to build a BOW model, people need to provide two things: the vocabulary of known words

and a measure of the presence of known words. Given the document collection $D = \{d_i,\ i = 1, 2, 3...n\}$ and m unique words in these documents, mathematically, each document d_i is represented by an $m \times 1$ vector $v_i \in R^{m\times 1}$. For instance, consider there are three documents in this collection D:

d_1: I like learning text mining.
d_2: What is text mining?
d_3: Apple tastes good.

Now we make a list of all words in our model's vocabulary. The unique words here (ignoring case and punctuation) are: {*I, like, learning, text, mining, what, is, apple, taste, good*}. Thus we have 10 unique words and 12 words in total within this collection D.

Next step is to score each word in the document. There are many methods of scoring. Let us consider the simple Boolean first. If a word appears in a document, its corresponding weight is 1; otherwise, it is 0. Since our vocabulary has 10 words, we use a fixed-length vector representation, with each position in the vector to score a word. Then the vector representations of these three documents are like this

$$v_1 = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0],$$

$$v_2 = [0, 0, 0, 1, 1, 1, 1, 0, 0, 0],$$

$$v_3 = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1].$$

Notice that the order of word index is the same as the unique word list above.

The intuition of this model is that the information within a document is from its content, which are words in this case. Documents are similar if they have similar words. Since each of the three vectors has a fixed length, we use cosine similarity to measure their similarity. Consider two vectors $\mathbf{A}$ and $\mathbf{B}$ with a fixed length N, and cosine similarity is defined as follows:

$$Cosine\ Similarity = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{i=N} A_i B_i}{\sqrt{\sum_{i=1}^{i=N} A_i^2}\sqrt{\sum_{i=1}^{i=N} B_i^2}}, \tag{4}$$

where A_i and B_i are components of vectors $\mathbf{A}$ and $\mathbf{B}$, respectively.

The cosine value ranges between $[-1, 1]$, 1 for vectors pointing at the same direction, 0 for orthogonal, and -1 for vectors pointing in the opposite direction. For documents, the term values are usually non-negative, so the cosine similarity ranges between [0, 1], and the higher the value is, the more similar two documents are.

Now we calculate the similarity between documents d_1, d_2 and d_3 using this formula,

$$\cos(d_1, d_2) = 0.4472;\ \cos(d_1, d_3) = 0;\ \cos(d_2, d_3) = 0.$$

We believe that this result is consistent with our observation, d_1 and d_2 are similar to each other because they are both talking about text mining, d_3 has no relation with d_1 and d_2, and thus their cosine similarity is zero.

The BOW model is very straightforward and is easy to implement. For the word weight in the BOW model, besides the simple Boolean model, we also use counts of words, frequency, or term frequency-inverse document frequency (tf-idf) as word weight, and more information about this is referred to Salton and Buckley (1988).

Despite the simplicity of this representation method, we face two significant disadvantages if we want to adapt this method on our comments data: (1) The vocabulary size in comments data is about 10,000, while each comment has only 10–200 words, and using the BOW model will lead to high-dimensional and sparse vectors. (2) BOW representation does not consider the semantic relation between words, and it assumes all the words are independent. This assumption may have some problems, for example, "used bicycle" and "old bike" will be considered as entirely different phrases because they have no words in common.

Next we develop a new text representation method based on the BOW model to overcome these disadvantages. First, we introduce word embeddings, a learned representation of words where words with the same meaning will have similar representations (Manning et al. 2008). Word2vec (see Mikolov et al. 2013a,b) is a very effective algorithm to train word embeddings based on the local documents. With these word embeddings, we group similar words as a cluster using the clustering method. Finally, instead of representing document (comment) as "bag of words," we represent them as "bag of word clusters." In this way, we retrieve semantic information from the text, for example, "bike" and "bicycle" are grouped together because they have the same or similar meaning. Moreover, the BOW model's sparsity problem is handled since the number of clusters is significantly smaller than the vocabulary size.

Word2vec was created and published in 2013 by a team of researchers led by Tomas Mikolov at Google (Mikolov et al. 2013a,b). Word2vec is a group of related models used to produce word vectors (also called word embeddings). Usually, word2vec is referred to two model architectures and two related training techniques:

- **2 model architectures**: continuous bag-of-words (CBOW) and skip-gram (SG). CBOW aims to predict a center word from the surrounding context in terms of word vectors. Skip-gram does the opposite and predicts the probability of context words from a center word.
- **2 training techniques**: negative sampling and hierarchical softmax. Negative sampling defines an objective by sampling negative examples, while hierarchical softmax defines an objective using an efficient tree structure to compute probabilities of appearance for all the vocabulary.

In our application, the skip-gram model with negative sampling is used. The detailed steps are given in the first author's master thesis. For a detailed explanation of other models in word2vec, one refers to Rong (2014). Moreover, since skip-gram is a neural network model, if you are not familiar with the neural network model, you can refer to Goodfellow et al. (2016).

```
w2v_model.wv.most_similar('baby')
```

```
[('child', 0.6217172145843506),
 ('babe', 0.5719802379608154),
 ('infant', 0.5226951241493225),
 ('little_guy', 0.5223316550254822),
 ('son', 0.5125443935394287),
 ('kid', 0.5019221305847168),
 ('kiddo', 0.4887546896934509),
 ('daughter', 0.47309964895248413),
 ('newborn', 0.4703175127506256),
 ('him', 0.4476194977760315)]
```

```
w2v_model.wv.most_similar('apple')
```

```
[('carrot', 0.5864819288253784),
 ('banana', 0.5851048231124878),
 ('fruit', 0.5847668051719666),
 ('veggie', 0.5716378688812256),
 ('grape', 0.5633351802825928),
 ('pear', 0.5447382926940918),
 ('strawberry', 0.5204207897186279),
 ('pea', 0.4928736686706543),
 ('avocado', 0.4899260997772217),
 ('fruit_veggie', 0.48924189805984497)]
```

```
w2v_model.wv.most_similar('well') # :
```

```
[('nicely', 0.6807569265365601),
 ('beautifully', 0.6258641481399536),
 ('wonderfully', 0.5551607608795166),
 ('fine', 0.5284466743469238),
 ('amazingly', 0.5153716802597046),
 ('too', 0.5069817304611206),
 ('perfectly', 0.5052441358566284),
 ('decently', 0.48130089044570923),
 ('sturdily', 0.46339577436447144),
 ('fabulously', 0.4559105634689331)]
```

Fig. 1 Word2vec example

Training using word2vec model produces N-dimensional vectors for each word in our vocabulary. These word embeddings have many good properties. Since each word embeddings have the same size N, it is easy to measure the distance between a pair of word embeddings. Another property of these pre-trained word embeddings is that semantically related words usually have a close distance. Here we use the cosine similarity defined by (4), and the more similar two words are the higher cosine similarity of their word embeddings. Figure 1 shows three examples of words "baby," "apple," and "well" with their top 10 most similar words in the whole vocabulary with around 10^4 words. More detail about how these word embeddings trained is described in Sect. 4.

In our method, we adopt the K-means algorithm (Hartigan & Wong 1979) to perform word embeddings clustering. K-means is a straightforward and efficient algorithm for general clustering. We introduce how K-means are applied in our method, and let us review this algorithm first.

In this clustering problem, we are given n word embeddings as our training set $\{w^{(1)}, w^{(2)}, ..., w^{(n)}\}$ and $w^{(i)} \in \mathbb{R}^N$. The number of clusters is a pre-set parameter K, and the K-means algorithm is as follows:

1. Initialize cluster centroids $\mu_1, \mu_2, ..., \mu_K \in \mathbb{R}^N$ randomly.
2. Repeat until convergence: {
   for every $i \in \{1, ..., n\}$, set

$$c^{(i)} := \arg\min_j ||w^{(i)} - \mu_j||^2; \tag{5}$$

for every $j \in \{1, ..., K\}$, set

$$\mu_j = \frac{\sum_{i=1}^{n} 1\{c^{(i)} = j\} w^{(i)}}{\sum_{i=1}^{n} 1\{c^{(i)} = j\}}. \tag{6}$$

}

For every repetition, there are two steps, first is to assign each training sample $w^{(i)}$ to its nearest cluster μ_j and update its assigned cluster index c^i. Then, update the cluster μ_j to the mean of the points assigned to it. The K-means algorithm is also regarded as a coordinate descent on the distortion function J,

$$J(c, \mu) = \sum_{i=1}^{n} ||w^{(i)} - \mu_{c^{(i)}}||^2. \tag{7}$$

Clearly, the distortion function is a non-convex function, so the K-means algorithm is easily got stuck in local minima. One common solution to this problem is to run K-means many times with a different random initialization of μ, and out of all different clusters founded, use the one with the lowest distortion J as our final solution.

Normally, we use cosine similarity to measure the distance between word embeddings as we mentioned before, but K-means use only Euclidean distance as the distance measure. Although other clustering methods use cosine as a distance measure like K-medoids (Kaufmann 1987), we still use K-means due to its much higher computational efficiency. And we justify that for normalized vectors, cosine similarity and Euclidean distance are linearly connected. For two normalized vectors $A = \{A_i\}$, $B = \{B_i\}$ ($\sum A_i^2 = \sum B_i^2 = 1$), the Euclidean distance between A and B is

$$\begin{aligned} ||A - B||^2 &= \sum (A_i - B_i)^2 \\ &= \sum (A_i^2 + B_i^2 - 2A_i B_i) \\ &= \sum A_i^2 + \sum B_i^2 - 2\sum A_i B_i \\ &= 1 + 1 - 2\cos(A, B) \\ &= 2(1 - \cos(A, B)). \end{aligned} \tag{8}$$

Note that for normalized vectors $\cos(A, B) = \frac{\sum A_i B_i}{\sqrt{\sum A_i^2}\sqrt{\sum B_i^2}} = \sum A_i B_i$. The higher two word embeddings' cosine similarity is, the closer their Euclidean

distance is, which is consistent with our objective. Thus in our application, we perform K-means on the normalized pre-trained word embeddings.

So instead of representing text documents as "bag of words," we represent them as "bag of word clusters." Now we have our pre-trained word embeddings, and then we perform K-means algorithm, which assigns each word a unique cluster index. Then constructing bag-of-word clusters representation of text document is summarized as the following steps:

1. Preprocess and tokenize the text, and then each text is represented as a list of words.
2. Given pre-trained K word cluster, replace each word in the list as its cluster index, if there are unknown words, replace them with $K + 1$, so this vector is transformed to numerical lists with the number 1 to $K + 1$.
3. Calculate each cluster's frequency in the text list, and construct a vector with length k+1 where each term will be the calculated frequency of clusters with the corresponding index number. And this new vector is our "bag of clusters" representation of text.

With the above steps, we transform each text into a $K + 1$-dimensional vector. For example, we have a short text:

```
"I love eating apples, they are delicious"
```

and we have four pre-trained word clusters: C_1 ={*"I," "they," "you"*}, C_2 ={*"apple," "pear," "banana"*}, C_3 = {*"is," "are," "was"*}, and C_4 = {*"delicious," "good," "tasty"*}. First, we tokenize this text as a vector v = [*"I," "love," "eating," "apples," "they," "are," "delicious"*], then replace these words with the corresponding clusters $v = [1, 5, 5, 2, 1, 3, 4]$, and remember to replace unknown words with "k+1" that is 5 here. Next we calculate each cluster's relative frequency: $f_1 = \frac{2}{7}$, $f_2 = \frac{1}{7}$, $f_3 = \frac{1}{7}$, $f_4 = \frac{1}{7}$, $f_5 = \frac{2}{7}$, and represent this text with new vector $v' = [f_1, f_2, f_3, f_4, f_5] = [\frac{2}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{2}{7}]$.

Text digitalization or representing text as numerical vectors is an essential part of every text mining application. Our "bag of word clusters" model can extract not only statistical information but also part of semantic information from text.

3 Ranking Comments with General Entropy

In this section, we use **General Entropy** (Zhang 2019) to rank comments based on the entropy value. First we define the **Maximum Entropy Comment**. By treating the maximum entropy comment as an "ideal" comment, we measure each comments distance to the "ideal" comment by using K-L divergence and rank comments based on its value. Moreover, there are two features of our comment ranking algorithm:

(1) Our method is unsupervised, which means there is no human-labeled training set to learn from, and all we have is a group of comments without any order. In other words, our method does not depend on an annotated training set.
(2) Judging the quality of a comment is subjective, and we cannot just create a judgment standard from nothing. So the objective of our method is not to distinguish these top-ranked comments but to make sure those unrelated or "fake" comments have as lower ranks as possible. In general, one of the objectives of our method is to filter out "bad" comments.

After the pre-trained word embedding clustering, given n clusters of words and m comments under a product, we regard the collection of all m comments together as the **Global Comments Set**. Then we calculate the number of each word cluster appears in the global comments set, which is represented as $\{Num_0^G, Num_1^G, Num_2^G, ..., Num_n^G\}$, and notice that Num_0^G is the number of unknown words that appear in the collection. Now we define the **global probability** of word cluster i in the global comments set as

$$Q_i = \frac{Num_i^G}{Num_0^G + Num_1^G + Num_2^G + ... + Num_n^G}, i = 0, 1, \ldots, n. \tag{9}$$

And for all global probabilities, we have

$$\sum_{i=0}^{n} Q_i = 1. \tag{10}$$

Since word2vec model has to be trained on a large corpus with around 10^4–10^5 unique words, we group these words into n word clusters. One feature of our bag-of-word clusters model is that these pre-trained word clusters are used for many products at the same time. So for one product, it is possible that no word within its global comments set falls into the word cluster i^*. In other words, it is possible that global probability $Q_{i^*} = 0$ for this product.

In our comments ranking method, we tend to treat each text or comment as a distribution of word clusters. If two comments have similar distributions, they probably expressed similar meanings. And as an unsupervised method, without any training set, the global comments set's distribution can be an essential reference for determining each individual comment's relevance to others. We believe that under a product, most comments will focus on some specific aspects of this product, which tend to have similar distributions of words, and if a comment has a completely different word distribution with the others, it might be a "fake comment."

Similarly, for an individual comment with index j, we have the number of each word cluster in the comment as $\{Num_0^j, Num_1^j, Num_2^j, ..., Num_n^j\}$, the probability of word cluster i in the jth comment is defined as

$$P_i^j = \frac{Num_i^j}{Num_0^j + Num_1^j + Num_2^j + ... + Num_n^j}, i = 0, 1 \ldots, n, \tag{11}$$

where

$$\sum_{i=0}^{n} P_i^j = 1, \tag{12}$$

and note that if $Q_i = 0$, then $P_i^j = 0$.

Thus with our new definition, for global comment set, the vector is $[Q_0, Q_1, Q_2, \ldots, Q_n]$, and for individual comment j is $[P_0^j, P_1^j, P_2^j, ..., P_n^j]$. By treating each comment as a distribution, we assess each comment's information quality by calculating entropy based on these probabilities.

In terms of comments, we treat each comment as a multinomial distribution of word clusters with probability $\mathbf{P}^j = [P_0^j, P_1^j, P_2^j, ..., P_n^j]$, that is, if we randomly sample a word from this comment, this word should have this probability distribution. For the worst scenario, if a comment only has one type of word in it like "good good good...good," then this comment has a distribution with $P(good) = 1$, and the entropy of this comment is zero. For the best scenario, without any constraint, the uniform distribution is the maximum entropy probability distribution for a random variable. The reason is that the entropy score is the "expected information gain," and the hardest distribution to predict is the uniform distribution when using a binomial score. For example, if a comment has an equal probability of every word cluster in it, it would have the maximum entropy. However, if we use entropy defined at (11) as our ranking score, a comment with uniform distribution would rank highest under any product, which cannot be used in our application. That is why we have to consider each comment relevance to the others, so we define the **General Entropy** as follows:

Definition 1 (General Entropy) Given global probability $\mathbf{Q} = \{Q_0, Q_1, ..., Q_n\}$ and a comment with probability $\mathbf{P}^j = \{P_0^j, P_1^j, ..., P_n^j\}$, the general entropy of this comment is

$$E(\mathbf{P}^j) = -\sum_{i=0, P_i^j \neq 0}^{n} Q_i \cdot P_i^j \cdot \log P_i^j.$$

Notice that P_i^j can be 0 since a comment may not include all word clusters. The general entropy measures the average information rate for an individual comment j with respect to the global probability. From the entropy definition given by (1), it assigns weight on each self-information of word cluster where the weight is the corresponding global probability. As we mentioned before, the global probability is regarded as a high-level abstraction of the topic in the comments under this product. Since we want to measure the information richness and the relevance of a comment,

we give higher weight to these words that other comments also mentioned and lower weight to words that other comments hardly mentioned.

At last, we summarize our general entropy ranking algorithm as follows:

Algorithm 1 Ranking comments based on the general entropy

Input:
Input: The set of n word clusters;
The set of m comments under a product;
Output:
Output Ranking results of all comments;
1: Covert all comments into their bag-of-word clusters representations;
2: Calculate the global probability $\mathbf{Q} = [Q_0, Q_1, Q_2, ..., Q_n]$;
3: Calculate each comment's probability: $\mathbf{P}^j = [P_0^j, P_1^j, P_2^j, ..., P_n^j],\ j = 1, 2...m$;
4: Calculate each comment's general entropy $E(\mathbf{P}^j)$;
5: Rank comments based on their general entropy, comment with higher general entropy is ranked higher;
6: **return** Ranking results;

In our experiment, comment with high complexity (e.g., very long comment with many different kinds of words) and almost no relevance to this product can get pretty high general entropy. These comments may have high ranks since most comments' entropy scores are close to each other. So instead of calculating scores for every comment, we first find an "ideal" comment and then judge comment by how far or how different it is from our "ideal" comment. Naturally, can define a comment with maximum general entropy as our "ideal" comment, and we call this "ideal" comment the **Maximum General Entropy Comment**. Since the maximum general entropy comment has the maximum general entropy, it keeps a good balance of relevance and information richness. We define the **Maximum General Entropy Comment** as follows:

Definition 2 (Maximum General Entropy Comment) Given global probability $\mathbf{Q} = \{Q_0, Q_1, \ldots, Q_n\}$, the maximum general entropy comment $\mathbf{B} := \{B_0, B_1, ..., B_n\}$ is defined as

$$\mathbf{B} = \arg\max_{\mathbf{P}} E(\mathbf{P}),$$

where $\mathbf{P} := \{P_0, P_1, ..., P_n\}$ and $\sum_{i=0}^{n} P_i = 1$. □

Note that the maximum entropy comment is a comment with the maximum general entropy within all possible comments. This comment may not exist in the existing comment set. However, it is regarded as a standard to judge each comment's relevance to the product. The following theorem shows that the maximum entropy comment exists and is unique, given a collection of comments. Its proof can be found in the first author's master thesis.

Theorem 1 *Given global probability* $\mathbf{Q} = \{Q_0, Q_1, ..., Q_n\}$ *and an index set* C *that* $i \in C$ *if* $Q_i \neq 0$ *and* $i \notin C$ *otherwise. Then there exists a unique maximum general entropy answer* $\mathbf{B} = \{B_0, B_1, ..., B_n\}$ *so that* $\mathbf{B} = \arg\max_{\mathbf{P}} E(\mathbf{P})$ *and* $B_i = \begin{cases} e^{-1-\frac{\lambda}{Q_i}} & i \in C \\ 0 & i \notin C, \end{cases}$ *where* λ *is a unique value and* $\sum_{i=0}^{n} B_i = 1$.

After the definition of the "ideal" comment, now we need a method to measure each comment's distance to the maximum general entropy comment. As we mentioned before, we treat each comment as a multinomial distribution with probability $\{P_0^j, P_1^j, ..., P_n^j\}$, that is, if we randomly sample a word from this comment, this word belongs to word cluster i with probability P_i^j. We treat the maximum general entropy answer the same way as it is a multinomial distribution with $\{B_0^j, B_1^j, ..., B_n^j\}$. To measure how one probability distribution is different from others, we use **K-L divergence** defined as follows:

Definition 3 (K-L Divergence) Given jth comment probability $\mathbf{P}^j = \{P_0^j, P_1^j, ..., P_n^j\}$ and the maximum general entropy comment $\mathbf{B} = \{B_0, B_1, ..., B_n\}$, the Kullback–Leibler divergence from the maximum general entropy comment $\mathbf{B}$ to jth comment $\mathbf{P}^j$ is defined to be

$$D_{KL}(\mathbf{P}^j|\mathbf{B}) = \sum_{i=0, P_i^j \neq 0}^{n} P_i^j \cdot \log\left(\frac{P_i^j}{B_i}\right).$$

In statistics, we call $\mathbf{B}$ the prior probability distribution and $\mathbf{P}^j$ the posterior probability distribution, and the K-L divergence from $\mathbf{B}$ to $\mathbf{P}^j$ is

$$\begin{aligned} D_{KL}(\mathbf{P}^j|\mathbf{B}) &= \sum_{i=0, P_i^j \neq 0}^{n} P_i^j \cdot \log\left(\frac{P_i^j}{B_i}\right) \\ &= -\sum_{i=1, P_i^j \neq 0}^{n} P_i^j \cdot \log(B_i) - \left(-\sum_{i=1, P_i^j \neq 0}^{n} P_i^j \cdot \log(P_i^j)\right) \qquad (13) \\ &= -\sum_{i=1, P_i^j \neq 0}^{n} P_i^j \cdot \log(B_i) - H(\mathbf{P^j}). \end{aligned}$$

$D_{KL}(\mathbf{P}^j|\mathbf{B})$ is actually the information gain if we use distribution $\mathbf{B}$ to approximate $\mathbf{P}^j$. When two distributions are close to each other, this value is relatively small, and otherwise, it is large if two distributions are very different. The first item $-\sum_{i=1, P_i^j \neq 0}^{n} P_i^j \cdot \log(B_i)$ in (13) is called **cross-entropy**, which is a very popular loss function of classification problem in machine-learning area.

Finally, we summarize our ranking process as follows:

Algorithm 2 Ranking comments based on K-L divergence to the maximum general entropy comment

Input:
The set of n word clusters;
The set of m comments under a product;
Output:
Output Ranking results of all comments;
1: Covert all comments into their bag-of-word clusters representations;
2: Calculate the global probability: $\mathbf{Q} = [Q_0, Q_1, Q_2, ..., Q_n]$;
3: Calculate each comment's probability: $\mathbf{P}^j = [P_0^j, P_1^j, P_2^j, ..., P_n^j],\ j = 1, 2...m$;
4: Based on the global probability $\mathbf{Q}$, find the maximum general entropy comment: $\mathbf{B} = \{B_0, B_1, ..., B_n\}$;
5: Calculate each comment's K-L divergence to the maximum general entropy comment $\mathbf{B}$;
6: Rank comments based on their K-L divergence, comment with lower divergence is ranked higher;
7: **return** Ranking results.

The next step is to assess this algorithm by evaluating the ranking quality. Here we introduce **normalized Discounted Cumulative Gain (nDCG)** (Järvelin & Kekäläinen 2002); it is often used to measure the effectiveness of web search engine algorithms, but it can also be applied to text ranking application. Many researches mentioned before (Hsu et al. 2009; Woloszyn et al. 2017) adapt this method to assess their ranking algorithm. First, let us define **Discounted Cumulative Gain (DCG)**.

Definition 4 (DCG) Given a ranked list with m comments, and rel_i is graded relevance of the result at position i, discounted cumulative gain is defined as

$$DCG_m = \sum_{i=1}^{m} \frac{rel_i}{\log_2(i+1)}.$$

According to this definition, if a comment with high graded relevance appears lower in the ranking result, it will be penalized as the graded relevance value is reduced logarithmically proportional to the position of the ranking result. To achieve high DCG value, the algorithm should rank a high relevance comment higher than low relevance one. Notice that in our application, graded relevance rel_i is a manually annotated comment quality score that will not be used as an input in our algorithm, and more detail about our experiment is described in Sect. 4.

While DCG is already a valid measure of ranking quality, it does not have a proper upper and lower bound to let people better compare the performance of different ranking results, and then **nDCG** is defined as follows:

Definition 5 (nDCG) Given a ranked list with m comments and its DCG value, the normalized discounted cumulative gain is computed as

$$nDCG_m = \frac{DCG_m}{IDCG_m},$$

where $IDCG_m$ is the ideal discounted cumulative gain.

$IDCG_m$ is straightforward to compute where the ideal ranking result is to rank these comments directly based on their graded relevance. nDCG ranges from 0 to 1, while 0 will not be able to achieve, and the closer our nDCG is to 1, the better quality our result has.

4 Experiment with Amazon Review Data

In this section, we conduct our experiment based on the Amazon product dataset (see He and McAuley 2016; McAuley et al. 2015), which contains users' reviews on Amazon website spanning 1996–2014. We choose one of the Amazon products and rank its comments using both pure general entropy and K-L divergence to "ideal comment." By comparing these two methods on a real dataset, we understand each method's characteristics and how they distinguish "fake" comments from actual comments. Moreover, we also analyze the relationship between general entropy and K-L divergence.

Amazon product data contain more than one hundred million reviews over millions of products, and these reviews were grouped into different categories. In our experiment, we chose the category "baby," which includes 160,782 comments of 7701 products. Notice that the dataset is titled "5-core," which means each of the users and products has at least 5 comments. In that case, we assume that most of the comments in this dataset are reasonable.

As described in Sect. 2, in order to keep our word embedding's quality, we need a large amount of data to train our word2vec model. In that case, we combine all comment text in "baby" category as our corpus, which is used as an input to our word2vec model. The first step is data cleaning that is an essential part of every text mining application. We need to carefully remove all noise or unnecessary words in the text and keep as much information as possible. We performed our data cleaning process using a Python package called Gensim (Řehůřek & Sojka 2010). After cleaning the dataset, we feed the corpus to our word2vec model with K-means clustering. The detailed steps are described in the first author's master thesis.

Table 1 shows part of the results of our word clusters; notice that the "Semantic category" titles are manually assigned and not used in the ranking algorithm. We observe that in cluster #133, word "she" is in the same cluster as "baby." That is an interesting feature of the word2vec model; remember our dataset is the collection of all products' comments under the "baby" category. In these comments, "she" always indicates a "baby," where these two words have similar context words and thus have similar word embeddings. That is also why word "poorly" and "perfectly" are in the same cluster; despite the two words are antonyms, they have similar context

words in the corpus. This property does not affect our application since we only care about each comment's relevance, and criticism and praise of a product are both information-rich comments.

Assigning similar words with the same cluster number enables us to distinguish comments more accurately. With our word embedding clusters, we transform each comment into its "bag of word clusters" digital representation. Now we apply our ranking algorithm on a real product. We used a product called "OXO Tot Waterproof Silicone Roll Up Bib with Comfort-Fit Fabric Neck" (ASIN: B00D3TPGAO) (Oxo tot waterproof silicone roll up bib with comfort-fit fabric neck 2014). This product also belongs to the "baby" category, which means we will have no difficulties transforming comments into their bag-of-word clusters representation. The details of this product are shown in Fig. 2.

This product has 95 comments in total. We checked all comments and made sure that they are all related to the products. Let us first take a look at the word cluster distribution of all comments, which we also called global probability in Sect. 3. Figure 3 is the histogram of global word distribution, where the horizontal axis represents the cluster index, and the vertical axis represents frequency. The word clusters distribution is not very uniform, and some clusters have significantly higher frequencies than the others. Tables 2 and 3 show the top 3 most frequent and least frequent word clusters in the whole collection of comments. Obviously, one of the most frequent words under this product should be "bib," and its corresponding

Table 1 Examples of the word embedding clusters

Cluster#	Semantic category	Examples of clustered words
133	Baby	Baby, son, daughter, child, kid, she, little_guy, kiddo, babe,...
11	Automobile	Car, trunk, vehicle, drive, SUV, truck, Sedan, van, Ford,...
248	Food	Banana, apple, veggie, pea, chicken, meat, pasta, avocado,...
116	Adverb	Well, fine, perfectly, nicely, properly, beautifully, poorly,...
104	Media	Picture, movie, video, image, show, pic, visual, television,...

Fig. 2 Product detail (Oxo tot waterproof silicone roll up bib with comfort-fit fabric neck 2014)

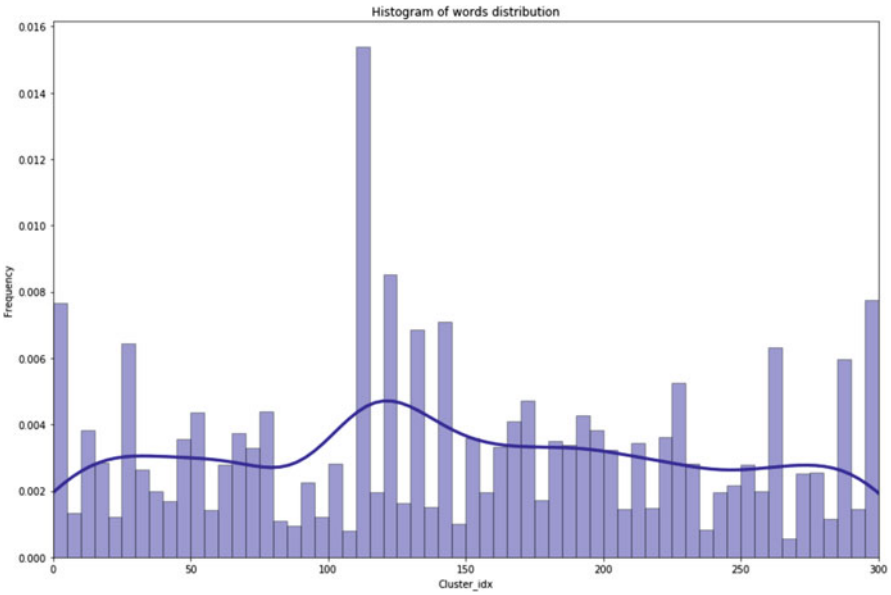

Fig. 3 Global word distribution

Table 2 Top 3 most frequent word clusters

Cluster#	Frequency	Examples of clustered words
112	0.0701	Bib, spoon, bowl, plate, dish, fork, catch_food...
133	0.0299	Baby, son, daughter, child, kid, little_guy, kiddo...
1	0.0216	Easy, easier, simple, useful, handy, make_easier...

word clusters have the highest frequency. It is not just because this cluster includes the word "bib," it also includes many related words such as "spoon," "fork," "catch_food," which may also appear a lot in the comments. The second most frequent cluster includes "baby" related words; these words appear a lot in the comments as this is a baby product. The third cluster is "easy" related words; many users describe this product using these adjectives. If these clusters have high frequency appearing in a comment, this comment is likely related to our product. After analyzing the global word cluster distribution, we find that global probabilities contain a lot of information regarding our product and are capable of judging the relevance of a comment to this product, which partially proves our methods' feasibility.

In the following, we apply our two ranking methods on our dataset: general entropy and K-L divergence to the maximum general entropy comment. In the dataset, we have one product with 95 comments, and these comments are considered as **relevant comment**. Since we do not judge these 95 comments' quality, we arbitrarily select 5 sets of comments that are not related to this product. Each set

Table 3 Top 3 least frequent word clusters

Cluster#	Frequency	Examples of clustered words
20	0	Phone, iPhone, tablet, computer, laptop, smartphone...
259	0	Brush, hair, bristle, toothbrush, nail_cliper, scalp...
263	0.0047	Sleep, sleeping, fall_asleep, cozy, snuggle, cuddle...

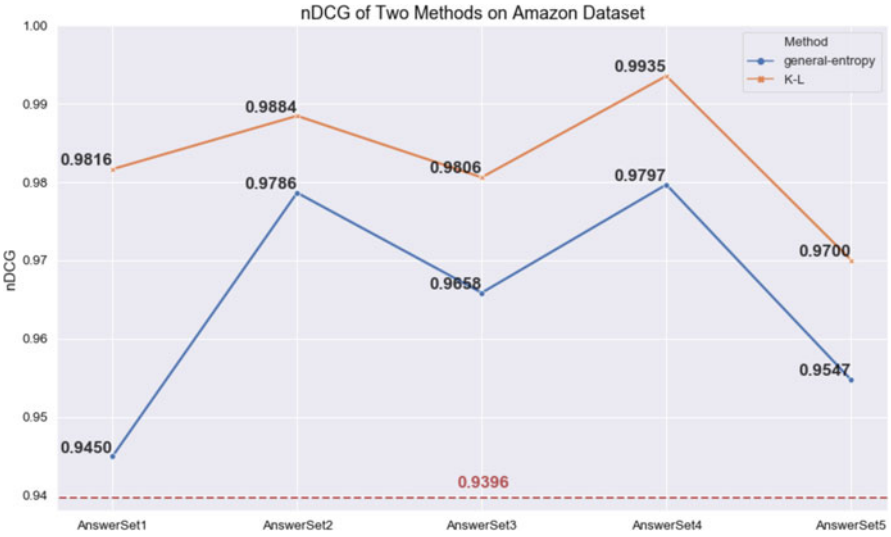

Fig. 4 nDCG of two methods on Amazon dataset

contains 10 comments. We called these comments **fake comment**, and they are all real comments under other Amazon products. It is worth mentioning that during the experiment, we are not aware of which comment is fake. We calculate global distribution and the maximum general entropy comment based on all comments, including fake comments.

To assess the ranking performance, we used the evaluation metric nDCG given in Sect. 3. According to the definition, we need to assign each comment a relevance score rel_i. In our experiment, we assigned the 95 original comments with relevance score **10** and fake comments with relevance score $-$**10**. To achieve higher nDCG value, the original comments should rank higher than fake comments. Moreover, in the best scenario, fake comments happen to have the lowest ranks, and we can calculate our ideal DCG (iDCG) based on this case.

Figure 4 shows the nDCG values of two methods on 5 different fake answer sets. Remember that nDCG ranges from 0 to 1, where a higher score indicates better performance. The red dashed line at the bottom of this figure is the baseline of our application. To construct the baseline, we generate 100,000 random sequences as the ranking results and calculate the mean of their nDCG values, which is 0.9396. Both of the two methods have better performance than the baseline, which shows our

methods' effectiveness. The K-L method generally outperforms the general entropy method, and they have the same trend. We observe that when the K-L achieves higher nDCG scores, the general entropy always has a higher score as well. The reason for this phenomenon is that they both rank comments based on their relevance to the global distribution. K-L method is more sensitive to each comment word distribution, and the variation of comments' K-L score is bigger than that of the general entropy method, so it has better discrimination power. In conclusion, the K-L method is more sensitive to the word distribution, while the general entropy puts more weight on the text complexity.

References

Alberto, T. C., Lochter, J. V., & Almeida, T. A. (2015). TubeSpam: Comment spam filtering on YouTube. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)* (pp. 138–143)

Chevalier, J. A., & Mayzlin, D. (2006). The effect of word of mouth on sales: Online book reviews. *Journal of Marketing Research, 43*(3), 345–354.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 28*(1), 100–108.

He, R., & McAuley, J. (2016). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th international conference on world wide web* (pp. 507–517).

Hsu, C., Khabiri, E., & Caverlee, J. (2009). Ranking comments on the social web. In *2009 International Conference on Computational Science and Engineering*, (vol. 4 , pp. 90–97).

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems, 20*(4), 422–446.

Kaufmann, L. (1987). Clustering by means of medoids. *Proceedings Statistical Data Analysis Based on the L1 Norm Conference, Neuchatel, 1987*, 405–416.

Manning, C. D., Schütze, H., & Raghavan, P. (2008). *Introduction to information retrieval*. USA: Cambridge University Press.

McAuley, J., Targett, C., Shi, Q., & Van Den Hengel, A. (2015). Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval* (pp. 43–52)

Mikolov, T., Chen, K., Corrado, G. S., & Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR abs/1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems 26*. Curran Associates, Inc. (pp. 3111–3119).

OXO tot waterproof silicone roll up bib with comfort-fit fabric neck. https://www.amazon.com/dp/B00D3TPGAO, 2014. Accessed: 2020-03-02.

Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 workshop on new challenges for NLP frameworks* (Valletta, Malta, May 2010). ELRA (pp. 45–50). http://is.muni.cz/publication/884893/en.

Rong, X. (2014). word2vec parameter learning explained. *CoRR abs/1411.2738*.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management 24*(5), 513–523.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal 27*(3), 379–423 (1948).

Woloszyn, V., dos Santos, H. D. P., Wives, L. K., & Becker, K. (2017). MRR: An unsupervised algorithm to rank reviews by relevance. In *Proceedings of the international conference on web intelligence*, WI'17 (pp. 877–883). New York, NY, USA: Association for Computing Machinery.

Zhang, G. (2019). How to rank answers in text mining. *Electronic Thesis and Dissertation Repository, 6250.* PHD Thesis, Western University.

Zhang, Z., & Varadarajan, B. (2006). Utility scoring of product reviews. In *Proceedings of the 15th ACM international conference on information and knowledge management*, CIKM '06 (pp. 51–57). New York, NY, USA: Association for Computing Machinery.