Theory: Intro to text representations

28 minutes 0 / 5 problems solved Skip this topic Start practicing

115 users solved this topic. Latest completion was about 2 hours ago.

Texts in natural language mainly consist of alphabetical characters, words, and bigger linguistic units. The question is, how can computers operate on such data? They apply various mathematical operations to compare numbers and find patterns in numeric input. So, if we want to do the same with the linguistic data, we should convert it into a numeric format, too.

The most convenient way is to provide a vector for each unit of a text. By **vector**, we mean an ordered sequence of numbers. In NLP, we can use them to describe any element, be it a single word or an entire text. What a vector encodes, depends solely on our objective and the approach we choose.

That's what the task of text representation deals with — how to convert a text into a vector. In this topic, we are going to cover several ways to do it.

Current topic:

Intro to text representations

Topic depends on:

Natural language processing

Topic is required for:

Overview of SpaCy

Word2Vec

Table of contents:

↑ Intro to text representations

§1. Bag-of-words model

§2. Other scoring methods

§3. Advantages and disadvantages of bag-of-words

§4. TF-IDF

§5. Example of calculating TF-IDF

§6. Word embeddings

§7. Conclusion

Feedback & Comments

§1. Bag-of-words model

Let's start with the **bag-of-words** representation, which is the easiest one to implement. Here we consider each word independently, without taking into account the surrounding context. We describe a text as a sequence of all words it contains, but we do not keep their original order and place (hence the name). The resulting vector encodes the text and stores information about occurrences of words in it. The model is frequently used in document classification and information retrieval but has applications in many other NLP tasks, too.

Let's look at the example. We have three reviews, each consisting of one sentence:

Review1: *easily the best album of the year.*

Review2: *the album is amazing.*

Review3: *loved the clean production!*

First, we need to design the **vocabulary**; it is the list of all known words across the data. If we ignore punctuation marks, it can look as follows: "*easily*", "*the*", "*best*", "*album*", "*of*", "*year*", "*is*", "*amazing*", "*loved*", "*clean*", "*production*". The vector length should equal the length of vocabulary, so it will be 11 here.

The next step is to count all occurrences of these words in each review. We can create a table, the columns of which will represent the units from the vocabulary:

	easily	*the*	*best*	*album*	*of*	*year*	*is*	*amazing*	*loved*	*clean*	*production*
Review1	1	2	1	1	1	1	0	0	0	0	0
Review2	0	1	0	1	0	0	1	1	0	0	0
Review3	0	1	0	0	0	0	0	0	1	1	1

For one, the article "*the*" appears twice in the first review, so we insert 2 in the corresponding cell opposite the document name. We obtain the following representations, in which a number in the vector represents the count of the corresponding word:

$$v_{Review1} = [1,2,1,1,1,1,0,0,0,0,0]$$

$$v_{Review2} = [0,1,0,1,0,0,1,1,0,0,0]$$

$$v_{Review3} = [0,1,0,0,0,0,0,0,1,1,1]$$

§2. Other scoring methods

There can be different ways of scoring. You can simply point out whether a word appears in a document or not. That leads to binary vectors, with 0 for each absent word and 1 for each present word. Now our representation will change a little:

$$v_{Review1} = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]$$

$$v_{Review2} = [0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0]$$

$$v_{Review3} = [0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1]$$

We create binary vectors when we are more concerned about the presence of words rather than their raw counts. The simplest sentiment analysis is an example of a task in which we can apply this representation.

Another approach is to calculate frequencies. You need to score occurrences of a particular word divided by the total number of words in a document. Let's illustrate it with "*the*": it appears twice in the first review which comprises 7 words overall. Hence, the result of their division is $\frac{2}{7}$. If we convert the number to a decimal fraction and round it up to two decimal places, we get 0.29. So, for all the reviews, we get these vectors:

$$v_{Review1} = [0.14, 0.29, 0.14, 0.14, 0.14, 0.14, 0.00, 0.00, 0.00, 0.00, 0.00]$$

$$v_{Review2} = [0.00, 0.25, 0.00, 0.25, 0.00, 0.00, 0.25, 0.25, 0.00, 0.00, 0.00]$$

$$v_{Review3} = [0.00, 0.25, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.25, 0.25, 0.25]$$

Counting frequencies makes sense when we have several documents. It is a way to compare the ratio of a specific word across the data, for instance, if one document consists of 25 words and the other one of 100 words. However, if you have only one text, raw counts will be enough to determine the most common terms.

§3. Advantages and disadvantages of bag-of-words

Now, we can name some advantages of the model:

- The main benefit lies in its simplicity. All values in a vector are easy to compute, and we can always tell what they stand for. In addition, despite the simplicity, it usually shows good performance in classification tasks.
- We can encode an entire text right away. If we do not need to pay attention to any of the inner structure, this approach is a good choice.
- We do not need a large dataset to build a model (as opposed to word embeddings you will learn about below).

However, there are some weaknesses, too:

- The model pays no attention to inner relations and neglects the word context, so the semantics is left out. Consider *buy* and *purchase;* in most cases, these words are synonyms and used in the same context, but we cannot access this information here.
- Often it provides sparse vectors with a huge amount of dimensions, that is vector length; in the bag-of-words it also equals the vocabulary length. Such vectors are both computation and memory consuming.

There is nothing we can do about the first disadvantage. As for the second one, standard preprocessing steps, such as text normalization and stopword removal, may help us. After the first procedure, various forms of one word ("*goes*", "*going*") will become a base form ("*go*"). This also applies to cases of "*Go*" and "*go*": if we do not convert "*Go*" to lowercase, these words will be recognized as two different units in the dictionary. After the second procedure, some high-frequency but meaningless words ("*a*", "*the*", "*are*", prepositions, etc.) will be removed completely. As a result, the vocabulary will include less items, and the vector will be shorter.

However, these steps are inefficient with large texts and dictionaries of thousands or millions of words. A bit later in this topic, we will observe another type of representation, which allows us to solve this problem.

§4. TF-IDF

Sometimes, when we score occurrences in the bag-of-words model, we get frequent words that are not important to a document we consider. Yet there can be words with smaller scores that are relevant to the topic of our text, but the bag-of-words does not pay enough attention to them. Here we have TF-IDF to help us increase proportionality and emphasize those words that actually contain useful information.

TF-IDF is short for *term frequency-inverse document frequency*. Here we mark counts not only in a specific document but also in an entire text collection. Some words appear in each document (for example, the verbs "*do*", "*say*", "*have*"), and their scores can be high even for one text. The idea of TF-IDF is to rescale the counts to get rid of such bias. That results in a weighted bag-of-words representation.

To get TF-IDF, you need to calculate two metrics:

- Term frequency captures the frequency of a word in one chosen document; it is the last method we showed in the previous section. Given some term w and a document d, we can calculate it as follows:

$$tf(w,d) = \frac{number\ of\ times\ w\ appears\ in\ d}{the\ total\ number\ of\ words\ in\ d}$$

- Inverse document frequency measures the rarity of a word across an entire set of documents. The more common a word, the closer its weight to 0. Given a document set D, the formula looks like this:

$$idf(w,D) = log\frac{the\ total\ number\ of\ documents\ in\ D}{the\ number\ of\ documents\ in\ D\ that\ include\ w}$$

The base of the logarithm can be different, but most commonly the natural logarithm is used. The TF-IDF score is the product of these two metrics:

$$tf\ idf(w) = tf(w,d) \times idf(w,D)$$

According to this representation, if a word is frequent in one document and not in others, it is valuable for this specific text. If the word occurs in most documents, it is regarded as meaningless and gets a very low weight due to IDF. In the end, we also get a vector representation, but this time the score for each word is calculated by means of TF-IDF.

§5. Example of calculating TF-IDF

Now, let's count the score for the article "*the*" from the first review that we introduced in the bag-of-words section:

$$tf = \frac{2}{7} \approx 0.29$$

The total number of documents is 3, so is the number of reviews that include this term. Here we take the natural logarithm:

$$idf = ln\,\frac{3}{3} = ln\,1 = 0$$

The result is 0, so the term is meaningless for the review, just as any other word that appears across the entire data:

$$tf\ idf = 0.29 \times 0 = 0$$

Let's provide vectors for all sentences:

$$v_{Review1} = [0.15, 0.00, 0.15, 0.06, 0.15, 0.15, 0.00, 0.00, 0.00, 0.00, 0.00]$$

$$v_{Review2} = [0.00, 0.00, 0.00, 0.10, 0.00, 0.00, 0.27, 0.27, 0.00, 0.00, 0.00]$$

$$v_{Review3} = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.27, 0.27, 0.27]$$

TF-IDF is an easy approach that can be intuitively understood. With its help, we can highlight the most valuable terms in a text. However, as it is based on the bag-of-words, we still ignore the grammar, semantics, and other linguistic aspects. The resulting vectors are high-dimensional and sparse, too. So, let's finally move on to a representation that can deal with these drawbacks.

§6. Word embeddings

Usually, we want our representation to keep the most of linguistic information from the data, such as semantic properties or word order.

For this purpose, we have a set of techniques known as **word embeddings**. Embeddings that they provide are also vectors, but they represent a word rather than the whole text, and the way its weights are calculated is not as obvious as in the previous approaches. Here we have lower-dimensional and denser vectors compared to the ones before, that means that they are shorter and have less zero values in them. In this case, they are easier to process.

Word embeddings allow us to capture semantic and syntactic relations. For example, the words "*lamp*" and "*lantern*" will have very similar vectors, while "*lantern*" and "*can*" will have different representations.

We cannot simply score the values manually, as we have done it with the bag-of-words and TF-IDF. Instead, the weights should be obtained during training on very large text collections. Alternatively, we can use pre-trained embeddings, for one, provided by the GloVe and Fasttest projects. The vector length can be set as a model parameter. Nowadays, there are two main types of word embeddings:

- **Context-independent**, where a model provides a single embedding, analyzing all usages of a word and generalizing them as one "meaning" in a vector. Besides GloVe that uses co-occurrence statistics, the well-known example is predictive word2vec.
- **Contextual**, where a model provides different word embeddings (different "meanings") based on the context of a word. ELMo and BERT are the most common examples.

As we noted, embeddings are usually given for a single word. In case you need to get one vector for a text, you can take the average, the sum, or the product of all word vectors in it. This is applied in NLP tasks, but you should keep in mind that once again you will miss the word order. Sometimes it will affect the semantics, too. Another approach is to weigh vectors with the TF-IDF scores first. As not all words represent the text meaning equally, we can decrease their importance. If we multiply each element in a vector by the corresponding TF-IDF value of a word, topic-relevant terms will have more weight, when determining the sense of the sentence.

In the next topics, you will learn more about some embedding models.

§7. Conclusion

In this topic, we overviewed some standard techniques for text representation. Summing up,

- Under the **bag-of-words**, we can use several metrics to describe the word occurrence, but the resulting vectors are usually high-dimensional, sparse, and can't keep additional linguistic information;
- **TF-IDF** is an advanced variant of the bag-of-words that measures the relevance of a word for a document. It has the same issues as the previous approach;
- **Embedding** models allow us to capture semantic relations between words and create low-dimensional dense vectors. Here words with resembling meanings tend to have similar vector representations.

Report a typo

12 users liked this theory. 0 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion