A Gentle Introduction to the Bag-of-Words Model

by **Jason Brownlee** on October 9, 2017 in **Deep Learning for Natural Language Processing**

The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms.
The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification.
In this tutorial, you will discover the bag-of-words model for feature extraction in natural language processing.

After completing this tutorial, you will know:

- What the bag-of-words model is and why it is needed to represent text.
- How to develop a bag-of-words model for a collection of documents.
- How to use different techniques to prepare a vocabulary and score words.

Tutorial Overview

This tutorial is divided into 6 parts; they are:

1. The Problem with Text
2. What is a Bag-of-Words?
3. Example of the Bag-of-Words Model
4. Managing Vocabulary
5. Scoring Words
6. Limitations of Bag-of-Words

The Problem with Text

A problem with modeling text is that it is messy, and techniques like machine learning algorithms prefer well defined fixed-length inputs and outputs.
Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.
In language processing, the vectors x are derived from textual data, in order to reflect various linguistic properties of the text.
This is called feature extraction or feature encoding.
A popular and simple method of feature extraction with text data is called the bag-of-words model of text.

What is a Bag-of-Words?

A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms.
The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.
A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

It is called a "*bag*" of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.
A very common feature extraction procedures for sentences and documents is the bag-of-words approach (BOW). In this approach, we look at the histogram of the words within the text, i.e. considering each word count as a feature.
The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document.
The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.
We will take a closer look at both of these concerns.

Example of the Bag-of-Words Model

Let's make the bag-of-words model concrete with a worked example.

Step 1: Collect Data

Below is a snippet of the first few lines of text from the book "A Tale of Two Cities" by Charles Dickens, taken from Project Gutenberg.
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,

For this small example, let's treat each line as a separate "document" and the 4 lines as our entire corpus of documents.

Step 2: Design the Vocabulary

Now we can make a list of all of the words in our model vocabulary.

The unique words here (ignoring case and punctuation) are:

- "it"
- "was"
- "the"
- "best"
- "of"
- "times"
- "worst"
- "age"
- "wisdom"
- "foolishness"

That is a vocabulary of 10 words from a corpus containing 24 words.

Step 3: Create Document Vectors

The next step is to score the words in each document.
The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.
Because we know the vocabulary has 10 words, we can use a fixed-length document representation of 10, with one position in the vector to score each word.
The simplest scoring method is to mark the presence of words as a boolean value, 0 for absent, 1 for present.
Using the arbitrary ordering of words listed above in our vocabulary, we can step through the first document ("*It was the best of times*") and convert it into a binary vector.
The scoring of the document would look as follows:

- "it" = 1
- "was" = 1
- "the" = 1
- "best" = 1
- "of" = 1
- "times" = 1
- "worst" = 0
- "age" = 0
- "wisdom" = 0
- "foolishness" = 0

As a binary vector, this would look as follows:

[1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

The other three documents would look as follows:

"it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]

"it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]

"it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

All ordering of the words is nominally discarded and we have a consistent way of extracting features from any document in our corpus, ready for use in modeling.
New documents that overlap with the vocabulary of known words, but may contain words outside of the vocabulary, can still be encoded, where only the occurrence of known words are scored and unknown words are ignored.
You can see how this might naturally scale to large vocabularies and larger documents.

Managing Vocabulary

As the vocabulary size increases, so does the vector representation of documents.
In the previous example, the length of the document vector is equal to the number of known words.
You can imagine that for a very large corpus, such as thousands of books, that the length of the vector might be thousands or millions of positions. Further, each document may contain very few of the known words in the vocabulary.
This results in a vector with lots of zero scores, called a sparse vector or sparse representation.
Sparse vectors require more memory and computational resources when modeling and the vast number of positions or dimensions can make the modeling process very challenging for traditional algorithms.
As such, there is pressure to decrease the size of the vocabulary when using a bag-of-words model.
There are simple text cleaning techniques that can be used as a first step, such as:

- Ignoring case
- Ignoring punctuation
- Ignoring frequent words that don't contain much information, called stop words, like "a," "of," etc.
- Fixing misspelled words.
- Reducing words to their stem (e.g. "play" from "playing") using stemming algorithms.

A more sophisticated approach is to create a vocabulary of grouped words. This both changes the scope of the vocabulary and allows the bag-of-words to capture a little bit more meaning from the document.

In this approach, each word or token is called a "gram". Creating a vocabulary of two-word pairs is, in turn, called a bigram model. Again, only the bigrams that appear in the corpus are modeled, not all possible bigrams.

An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (more commonly called a trigram) is a three-word sequence of words like "please turn your", or "turn your homework".

For example, the bigrams in the first line of text in the previous section: "It was the best of times" are as follows:

- "it was"
- "was the"
- "the best"
- "best of"
- "of times"

A vocabulary then tracks triplets of words is called a trigram model and the general approach is called the n-gram model, where n refers to the number of grouped words.

Often a simple bigram approach is better than a 1-gram bag-of-words model for tasks like documentation classification.

a bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat.

Scoring Words

Once a vocabulary has been chosen, the occurrence of words in example documents needs to be scored.

In the worked example, we have already seen one very simple approach to scoring: a binary scoring of the presence or absence of words.

Some additional simple scoring methods include:

- **Counts**. Count the number of times each word appears in a document.
- **Frequencies**. Calculate the frequency that each word appears in a document out of all the words in the document.

Word Hashing

You may remember from computer science that a hash function is a bit of math that maps data to a fixed size set of numbers.

For example, we use them in hash tables when programming where perhaps names are converted to numbers for fast lookup.

We can use a hash representation of known words in our vocabulary. This addresses the problem of having a very large vocabulary for a large text corpus because we can choose the size of the hash space, which is in turn the size of the vector representation of the document.

Words are hashed deterministically to the same integer index in the target hash space. A binary score or count can then be used to score the word.

This is called the "*hash trick*" or "*feature hashing*".

The challenge is to choose a hash space to accommodate the chosen vocabulary size to minimize the probability of collisions and trade-off sparsity.

TF-IDF

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much "informational content" to the model as rarer but perhaps domain specific words.

One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like "the" that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

- **Term Frequency**: is a scoring of the frequency of the word in the current document.
- **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

The scores are a weighting where not all words are equally as important or interesting.

The scores have the effect of highlighting words that are distinct (contain useful information) in a given document.

Thus the idf of a rare term is high, whereas the idf of a frequent term is likely to be low.

Limitations of Bag-of-Words

The bag-of-words model is very simple to understand and implement and offers a lot of flexibility for customization on your specific text data.

It has been used with great success on prediction problems like language modeling and documentation classification.

Nevertheless, it suffers from some shortcomings, such as:

- **Vocabulary**: The vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations.
- **Sparsity**: Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons, where the challenge is for the models to harness so little information in such a large representational space.
- **Meaning**: Discarding word order ignores the context, and in turn meaning of words in the document (semantics). Context and meaning can offer a lot to the model, that if modeled could tell the difference between the same words differently arranged (“this is interesting” vs “is this interesting”), synonyms (“old bike” vs “used bike”), and much more.