Bag of words

When using machine learning techniques to model text, the bag-of-words model is one way to represent text data. Language modelling and document categorization have experienced considerable success with the bag-of-words approach since it is straightforward to grasp and use. Natural language processing uses the bag-of-words feature extraction paradigm, which you'll learn about in this article.

Let's begin.

Article Overview

This article 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

- Text is difficult to model because it is unstructured, and machine learning algorithms favour well-defined inputs and outputs with constant lengths.
- Text cannot be used directly by machine learning algorithms; it must be transformed into numbers first. Number vectors, to be precise.
- Vectors x are constructed from textual data in language processing to represent various linguistic aspects of the text.
- Features are extracted or encoded in this way. The bag-of-words model of text is a popular and straightforward feature extraction method for text data.

What is a Bag-of-Words?

It's called a "bag of words" model because it uses machine learning techniques to extract features from text.

There are numerous ways to extract information from documents using this methodology, which is both simple and adaptable.
It's a text representation that shows where words appear in a document. There are two components to this:

- A pre-existing vocabulary
- A count of the occurrences of well-known words.

As the name implies, this document is a "bag" of words with no regard for word order or organisation. Not where in the document, but whether or not known words appear in it is all that matters to the model.

The bag-of-words approach is a popular feature extraction method for sentences and texts (BOW). As a result of this strategy, we examine the text's histogram of words, treating each word count as a feature.

If two documents contain the same content, the logical conclusion is that they are comparable. The document's meaning can also be deduced from the content alone.

Simple or complex word bags are both acceptable for the word bag. Choosing how to design the vocabulary of well-known words (or tokens) and determining how to rate the existence of well-known words are both difficult tasks.

Both of these issues will be investigated further.

Example of the Bag-of-Words Model

As an example, let's put the bag-of-words model into practise.

Step 1: Gather information.

Project Gutenberg provided the opening few lines of Charles Dickens' "A Tale of Two Cities" below.

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,

Each line will be treated as a "document" for the purposes of this example, and the four lines will represent our whole collection of documents.

Step 2: Developing the Vocabulary

The words in our model vocabulary are now ready for a list. If you ignore capitalization and punctuation, these are the only words you'll notice:

“it”
“was”
“the”
“best”
“of”
“times”
“worst”
“age”
“wisdom”
“foolishness”

That's 10 words out of a total of 24 in a corpus.

Step 3: Generate Documents Vectors

The next stage is to assign a score to each document based on the words within it. The goal is to convert each piece of free text into a vector that can be fed into or used as an output by a machine learning model.

Due to the fact that we know there are 10 words in the vocabulary, we may utilise a fixed-length document representation of 10 to score each word.

Using a boolean value to indicate the presence or absence of words is the most basic scoring approach. In our vocabulary, we have a list of phrases that we can use to create a binary vector from the first text ("It was the best of times").

The document will be scored in the following manner:

“it” = 1
“was” = 1
“the” = 1
“best” = 1
“of” = 1
“times” = 1

“worst” = 0
“age” = 0
“wisdom” = 0
“foolishness” = 0

This would appear as follows as a binary vector:
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

The following is the format for the other three documents:
"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]

With no regard to word order, we have a consistent method for extracting characteristics from each document in our corpus that can be used in model building. It is still possible to code new documents that contain terms that do not belong in the vocabulary, as long as just the occurrences of the vocabulary's well-known words are taken into account. Unknown words are ignored. It's easy to imagine how this would work with much larger vocabulary and document sizes.

Managing Vocabulary

Constantly expanding vocabulary means more vector representations of documents are needed. The document vector in the preceding example has a length equal to the number of recognised words.

For a big corpus, such as a library with thousands of books, the vector's length could be in the millions of positions range. Furthermore, only a small percentage of the vocabulary's recognised words may be found in any one document. There will be lots of zero scores in the result, hence the vector will be dubbed an "inefficient" one.

For standard techniques, sparse vectors require additional memory and processing resources because of the large number of places or dimensions. As a result, when utilising a bag-of-words paradigm, there is some pressure to reduce the vocabulary size.

As a beginning step, consider using simple text cleaning techniques like:

- Ignoring case
- Ignoring punctuation

- Ignoring frequent terms that carry little information, referred to as stop words, such as "a," "of," and so on.
- Correcting spelling and grammatical errors.
- Using stemming algorithms, words are reduced to their root (e.g. "play" from "playing").

Creating a vocabulary of grouped terms is a more advanced method. This broadens the vocabulary's scope while also giving the bag-of-words the opportunity to capture more of the document's meaning.

Using this method, each word or symbol is referred to as a "gramme". A bigram model is a method for creating a vocabulary made up of two-word pairs. This time around we are simply modelling the bigrams found in the corpus, not all conceivable bigrams.

"Please turn", "turn your", or "your assignment" are examples of 2-grams (often referred to as bigrams). A 3-gram (usually referred to as a trigram), on the other hand, is a three-word sequence of examples of 2-grams (commonly referred to as bigrams).

The bigrams in the previous section, for example, are as follows: "It was the best of times."

“it was”
“was the”
“the best”
“best of”
“of times”

The trigram model is a vocabulary's way of keeping track of triplets of words, while the n-gram model refers to the overall method. For situations like classification of documentation, a basic bigram approach often works better than a 1-gram bag of words model. Much more powerful than just words, a depiction made up of bigrams is extremely difficult to beat.

Scoring Words

Once a vocabulary list has been finalised, it's time to grade examples of those words in use.A simple approach to scoring was demonstrated in the worked example, which used a binary technique to score the presence or absence of words.In addition to these straightforward techniques of scoring, consider the following:

- **Counts:** Count the number of times a term appears in a document to get an idea of frequency.

- **Frequencies:** Measure how often each word appears in a text by looking at the total number of words present.

Word Hashing

A hash function, which you may recall from computer science, maps data to a defined set of numbers. When programming, we might utilise them in hash tables, where names are translated to numbers for quick retrieval.

Words in our lexicon can be represented as a hash using this technique. As a result, the problem of a big text corpus with a broad vocabulary is addressed since the size of the hash space may be customised to match the vector representation of the document.

Words are deterministically hashed in the target hash space to the same integer index. The word can then be graded using a binary score or count. The "hash trick" or "feature hashing" refers to this technique.

Choosing a hash space that accommodates the selected vocabulary size while minimising collisions and balancing sparsity is the difficult part.

TF-IDF

If you score word frequency, you'll have to deal with terms that are common but don't necessarily have as much "informational content" for the model as those that are uncommon, however more specialised (e.g. smaller scores).

An alternative strategy is to penalise words like "the" like "the" that occur often across all papers by rescaling the frequency of those words.It's called Term Frequency – Inverse Document Frequency (or TF-IDF for short), and it uses the following scoring method:

- Term Frequency: The frequency of a word in the current document is measured using the Term Frequency metric.
- Inverse Document Frequency: A word's inverse document frequency tells you how uncommon it is throughout all of the sources it appears in.

The scores are based on a grading system in which not all words are given the same weighting. It highlights words that are distinct (contain relevant information) in a document because of the scores assigned to them. As a result, the idf for a rare term is very high, but the idf for a frequent term is quite low.

Limitations of Bag-of-Words

You can customise your text data a lot using the bag-of-words concept because it's straightforward to learn and put into practise.
It's proved quite useful for problems like language modelling and documentation classification, where accuracy has been crucial.
Despite this, it has a few flaws, such as:

- **Vocabulary:** Managing the vocabulary's size has an impact on the sparsity of the document's representations, therefore it must be carefully designed.
- **Sparsity:** For computational (space and time complexity) as well as information reasons, sparse representations are more difficult to describe since the models must harness so little information in such a vast representational space.
- **Meaning:** Disregarding the order of words in a document overlooks context and, as a result, the meaning of those words (semantics). For example, context and meaning can help the model distinguish between synonyms ("this is fascinating" vs. "is this intriguing"), different word arrangements ("old bike" vs. "used bike," etc.).