About sklearn / scikit-learn :- <https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/>

<https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html>

Text data requires special preparation before you can start using it for predictive modeling.

The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).

The scikit-learn library offers easy-to-use tools to perform both tokenization and feature extraction of your text data.

In this tutorial, you will discover exactly how you can prepare your text data for predictive modeling in Python with scikit-learn.

After completing this tutorial, you will know:

* How to convert text to word count vectors with CountVectorizer.
* How to convert text to word frequency vectors with TfidfVectorizer.
* How to convert text to unique integers with HashingVectorizer.

Let’s get started.

**Bag-of-Words Model**

We cannot work with text directly when using machine learning algorithms.

Instead, we need to convert the text to numbers.

We may want to perform classification of documents, so each document is an “input” and a class label is the “output” for our predictive algorithm. Algorithms take vectors of numbers as input, therefore we need to convert documents to fixed-length vectors of numbers.

Vector meaning: - a quantity having direction as well as magnitude, especially as determining the position of one point in space relative to another.

A simple and effective model for thinking about text documents in machine learning is called the Bag-of-Words Model, or BoW.

The model is simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document.

This can be done by assigning each word a unique number. Then any document we see can be encoded as a fixed-length vector with the length of the vocabulary of known words. The value in each position in the vector could be filled with a count or frequency of each word in the encoded document.

This is the bag of words model, where we are only concerned with encoding schemes that represent what words are present or the degree to which they are present in encoded documents without any information about order.

There are many ways to extend this simple method, both by better clarifying what a “word” is and in defining what to encode about each word in the vector.

The scikit-learn library provides 3 different schemes that we can use, and we will briefly look at each.

## Word Counts with CountVectorizer

The [CountVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

You can use it as follows:

1. Create an instance of the CountVectorizer class.
2. Call the fit() function in order to learn a vocabulary from one or more documents
3. Call the transform() function on one or more documents as needed to encode each as a vector.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

Because these vectors will contain a lot of zeros, we call them sparse. Python provides an efficient way of handling sparse vectors in the [scipy.sparse](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html) package.

The vectors returned from a call to transform() will be sparse vectors, and you can transform them back to numpy arrays to look and better understand what is going on by calling the toarray() function.

Below is an example of using the CountVectorizer to tokenize, build a vocabulary, and then encode a document. Check about\_skleanr.py file which I build with below code.

from sklearn.feature\_extraction.text import CountVectorizer

# list of text documents

text = ["The quick brown fox jumped over the lazy dog."]

# create the transform

vectorizer = CountVectorizer()

# tokenize and build vocab

vectorizer.fit(text)

# summarize

print(vectorizer.vocabulary\_)

# encode document

vector = vectorizer.transform(text)

# summarize encoded vector

print(vector.shape)

print(type(vector))

print(vector.toarray())

with ‘print(vectorizer.vocabulary\_) ‘# output is {'the': 7, 'quick': 6, 'brown': 0, 'fox': 2, 'jumped': 3, 'over': 5, 'lazy': 4, 'dog': 1}

We can see that all words were made lowercase by default and that the punctuation was ignored. These and other aspects of tokenizing can be configured and I encourage you to review all of the options in the [API documentation](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html).

Running the example first prints the vocabulary, then the shape of the encoded document. We can see that there are 8 words in the vocab, and therefore encoded vectors have a length of 8. print(vector.shape) # output is (1, 8)

We can then see that the encoded vector is a sparse matrix ‘print(type(vector)) # output is <class 'scipy.sparse.csr.csr\_matrix'>’. Finally, we can see an array version of the encoded vector showing a count of 1 occurrence for each word except the (index and id 7) that has an occurrence of 2. print(vector.toarray()) # output is [[1 1 1 1 1 1 1 2]]

Importantly, the same vectorizer can be used on documents that contain words not included in the vocabulary. These words are ignored and no count is given in the resulting vector.

For example, below is an example of using the vectorizer above to encode a document with one word in the vocab and one word that is not.

# encode another document

text2 = ["the puppy"]

vector = vectorizer.transform(text2)

print(vector.toarray()) # output [[0 0 0 0 0 0 0 1]]

Running this example prints the array version of the encoded sparse vector showing one occurrence of the one word in the vocab and the other word not in the vocab completely ignored.

The encoded vectors can then be used directly with a machine learning algorithm.

## Word Frequencies with TfidfVectorizer

[TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf) This is an acronym than stands for “Term Frequency – Inverse Document Frequency”

Word counts(with CountVectorizer) are a good starting point, but are very basic.

One issue with simple counts is that some words like “the” will appear many times and their large counts will not be very meaningful in the encoded vectors.

An alternative is to calculate word frequencies, and by far the most popular method is called [TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf). This is an acronym than stands for “Term Frequency – Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

* **Term Frequency**: This summarizes how often a given word appears within a document.
* **Inverse Document Frequency**: This downscales words that appear a lot across documents.

Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

The [TfidfVectorizer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. Alternately, if you already have a learned CountVectorizer, you can use it with a [TfidfTransformer](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html) to just calculate the inverse document frequencies and start encoding documents.

The same create, fit, and transform process is used as with the CountVectorizer.

Below is an example of using the TfidfVectorizer to learn vocabulary and inverse document frequencies across 3 small documents and then encode one of those documents.

from sklearn.feature\_extraction.text import TfidfVectorizer

# list of text documents

text = ["The quick brown fox jumped over the lazy dog.",

"The dog.",

"The fox"]

# create the transform

vectorizer = TfidfVectorizer()

# tokenize and build vocab

vectorizer.fit(text)

# summarize

print(vectorizer.vocabulary\_) # output is {'the': 7, 'quick': 6, 'brown': 0, 'fox': 2, 'jumped': 3, 'over': 5, 'lazy': 4, 'dog': 1}

print(vectorizer.idf\_) # output is [1.69314718 1.28768207 1.28768207 1.69314718 1.69314718 1.69314718 1.69314718 1. ]

# encode document

vector = vectorizer.transform([text[0]])

# summarize encoded vector

print(vector.shape) # output is (1, 8)

print(vector.toarray()) # output is [[0.36388646 0.27674503 0.27674503 0.36388646 0.36388646 0.36388646 0.36388646 0.42983441]]

A vocabulary of 8 words is learned from the documents and each word is assigned a unique integer index in the output vector.

The inverse document frequencies are calculated for each word in the vocabulary, assigning the lowest score of 1.0 to the most frequently observed word: “the” at index 7.

Finally, the first document is encoded as an 8-element sparse array and we can review the final scorings of each word with different values for “the“, “fox“, and “dog” from the other words in the vocabulary.

The scores are normalized to values between 0 and 1 and the encoded document vectors can then be used directly with most machine learning algorithms.