DAT300 CHAPTER 8 – Applying Machine Learning to Sentiment Analysis

Jorid Holmen
Applied Robotics, Norwegian University of Life Sciences

This chapter is about **sentiment analysis**, which is a subfield of natural language processing (NLP). Sentiment analysis is about classifying documents based on the attitude of the writer (some times called opinion mining or polarity). A popular task in sentiment analysis is the classification of documents based on the expressed opinions or emotions of the authors with regard to a particular topic.

Contents

1 Introducing the bag-of-words model

Bag-of-words is a feature extraction technique that allows us to represent the text as numerical vectors. We create a vocabulary of unique tokens (for example words) from the entire set of documents. Then, we construct a feature vector from each document that contains the **counts** of how often each word occurs in the particular document. In other words, a bag-of-words model uses a vocabulary of known words, and a measure of the precense of the known words. Since the unique words from each document represent only a small subset of all the words in the bag-of-words vocabulary, the feature vectors will mostly consist of zeros, which is why we call them **sparse**.

To construct a bag-of-words model based on the word counts in the respective documents, we can use the `CountVectorize` class implemented in scikit-learn. It takes an array of text data, which can be documents or sentences, and constructs the bag-of-words model. In the bag-of-words model, the word or term order in a sentence or document does not matter. The order in which the term frequencies appear in the feature vector is dervied from the vocabulary indices, which are usually assigned alphabetically.

The following three sentences:

- 'The sun in shining'
- 'The weather is sweet'
- 'The sun is shining, the weather is sweet, and one and one is two'

is constructed into this vocabulary:

```
{'and': 0,
'two': 7,
'shining': 3,
'one': 2,
'sun': 4,
'weather': 8,
'the': 6,
'sweet': 5,
'is': 1}
```

which is transformed into the following sparse feature vectors:

```
[[0 1 0 1 1 0 1 0 0]
 [0 1 0 0 0 1 1 0 1]
 [2 3 2 1 1 1 2 1 1]]
```

The number in the vocabulary represent the index the word recieves in the feature vectors. Each indent in the feature vectors shown here corresponds to the integer values that are stored as dictionary items in the vocabulary. The

first feature at index one represents 'and', and the second word is 'is'. The rest follows in alphabetical order.

These values in the feature vectors are also called **raw term frequencies: *tf(t, d)*** - the number of times a term, t, occurs in a document, d.

1.1 n-grams and K-mers

The sequence of items in the bag-of-words model is also called 1-gram or unigram model, meaning that each item or token in the vocabulary represents a single word. The contiguous sequence of items in NLP are also called n-grams. The choice of the number n in the gram model depends on the particular application.

Counts of word sequences

- The sentence 'DAT300 is great and I love it' gives the 2-grams

  DAT300 is, is great, great and, and I, I love, love it

- The sentence: 'DAT300 is great and I love it' gives the 3-grams

  DAT300 is great, is great and, great and I, and I love, I love it

1.2 Assesing word relevancy via term frequency-inverse document frequency

When we are analyzing text data, we often encounter words that occur across multiple documents from both classes. These frequently occuring words typically do not contain useful information, and should therefore be downweighted. To downeight these frequently occuring words, there is a technicue called **term frequency-inverse document frequency (tf-idf)**. Tf-idf can be defined as the product of the term frequency and the inverse document frequency:

$$tf - idf(t,d) = tf(t,d) \times idf(t,d) \tag{1}$$

tf(t, d): term frequency
$idf(t,d)$: inverse document frequency, which can be calculated as follows:

$$idf(t,d) = \log\left(\frac{n_d}{1 + df(d,t)}\right) \tag{2}$$

n_d: the total number of documents
$df(d,t)$: the number of documents d that contain the term t. Note that adding the constant 1 to the denominator is optional and serves the purpose of assigning a non-zero value to terms that occur in none of the training examples
log: is applied to prevent low document frequencies from receiving too much weight

The IDF measures how rare or common a word is in the collection of documents. If a word appears in many documents, its IDF score will be lower, as this word is less unique to this certain document. If a word appears in fewer documents, its IDF score will be higher, as this word is more unique to this certain document.

A potential drawback of using TF-IDF for sentiment analysis on very short texts like tweets or SMS messages, is that the IDF part of TF-IDF may overemphasize rare terms which may not be relevant to sentiment. Another potential drawback is that the TF-IDF typically requires large amounts of text to accurately compute document frequencies.

1.2.1 scikit-learn implementation of TF-IDF

The scikit-learn implementation of TF-IDF differs slightly from the textbook version. It uses the following formulas:

Inverse Document Frequency ($idf(t, d)$):

$$\text{idf}(t, d) = \log \frac{1 + n_d}{1 + \text{df}(d, t)} \tag{3}$$

TF-IDF ($tf\text{-}idf(t, d)$):

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times (\text{idf}(t, d) + 1) \tag{4}$$

Tf values are typically normalized before computing TF-IDFs, but in scikit-learn, normalization is done afterward using L2 normalization per document. The normalization formula is given by:

$$v_{\text{norm}} = \frac{v}{||v||_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} = \frac{v}{\left(\sum_{i=1}^{n} v_i^2\right)^{\frac{1}{2}}} \tag{5}$$

1.2.2 Using scikit-learn Implementation of TF-IDF when `smooth_idf=False`

When the parameter `smooth_idf=False` in some scikit-learn implementations, the formula for $idf(t, d)$ is modified:

$$\text{idf}(t, d) = \log \left[\frac{n_d}{\text{df}(d, t)} \right] + 1 \tag{6}$$

The TF-IDF formula remains the same:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t, d) \tag{7}$$

Here, n_d represents the total number of documents, and $\text{df}(d, t)$ is the number of documents d that contain the term t. The addition of "1" in the *idf* equation ensures that terms with zero *idf*, i.e., terms that occur in all documents in a training set, will not be entirely ignored.

1.3 Cleaning the data

The first important step before we build out bag-of-words model is to clean the text data by stripping it of all unwanted characters. A text can, for example, contain HTML markups, puntuation or other non-letter characters. While HTML markups does not contain many useful semantics, puncuation marks can represent useful, additional information in certain NLP contexts.

1.4 Preprocessing documents into tokens

Preprocessing of documents into tokens are more relevant to machine translations. Raw text can be converted to words in several ways, and one way to *tokenize* documents is to split them into individual words by splitting the cleaned documents at their whitespace characters.

Another useful technique is **word stemming**, which is the process of transforming a word into its root form. It allows us to map related words to the same stem. There are different kinds of stemming, like porter stemming and snowball stemming.

Lemmetization is a technique in NLP that involves reducing words into theis base or dictionary form, for example 'better' becomes 'good'. Unlike stemming, which may produce a root form of a word that is not necessarily a valid word in the language, lemmetization ensures that the resulting word (called the lemma) is a valid word. Lemmetization is, however, computatonally expensive and it has been observed that it might not be that effective on text classification.

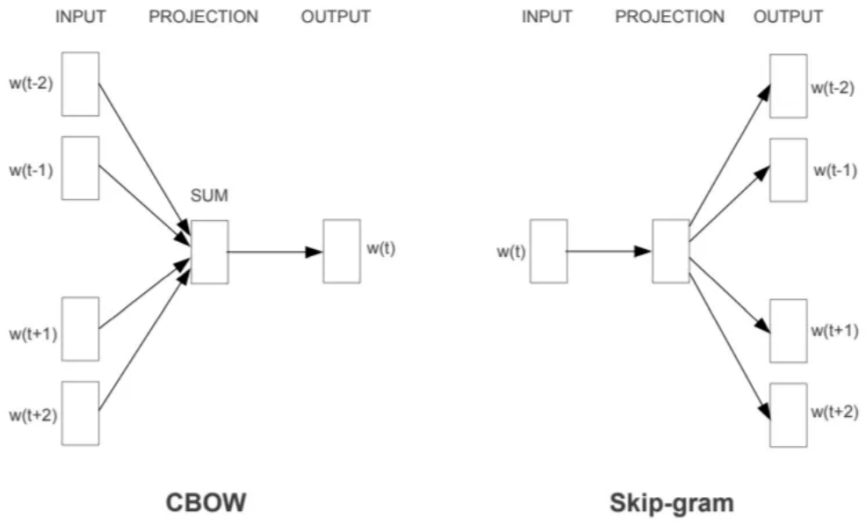
There is another useful topic called **stop-word removal**. Stop-words are simply those words that are extremely common in all sorts of texts and probably bear no useful information that can be used to distinguish between different classes in a document. Examples are *is, and, has, like*, etc. Removing stop-words can be useful if we are working with raw or normalized term frequencies rather than tf-idf, which are already downweighting frequently occuring words. Some types of language processing need the stop words, and here it is better to use stemming.

When using TF-IDF for sentiment analysis,it might be important to apply additional preprocessing steps such as stop-word removal or stemming, because TF-IDF relies on the lexical form of words and does not capture semantic meanings. However, TF-IDf are robust against stop-words.

2 Word2Vec

Word2Vec is a group of related models that transform words into vectors. It is an unsupervised learning algorithm based on neural networks that attempts to automatically learn the relationshsips between words by capturing the semantic meaning based on the context of words. The idea is to put words that have similar meanings into similar clusters, and via clever vector-spacing, the model can reproduce certain words using a simple vector math, for example *king - man + woman = queen.*

Traditional methods, like encoding and bag-of-words, reperesents words as discrete symbols, which leads to a very sparse, high-dimensional and hard-coded representation. We have a need for capturing semantic meaning in a continuous space, which can be solved using Word2vec, which gives more dense, lower-dimensional representation that learns from data.



2.1 Continuous Bag of Words (CBOW)

CBOW is a type of Word2Vec model. The goal is, given a context, to predict the target word. It takes in the context (surrounding words) as input and tries to predict the word in the middle. The idea is that the semantic meaning of a word can be inferred by the words surrounding it. For CBOW, the one-hot encoded context vectors are averaged or summed, passed through the hidden layer, and then the output layer to predict the target word.

For a sentence 'the cat sat on the mat', the context words are 'the', 'cat', 'on', and 'the', and the target word is 'sat'. CBOW tries to predict the target word based on the context.

2.2 Skip-Gram

Skip-Gram is another type of Word2Vec model. The goal is the predict the context given a word. It takes the word as input and tries to predict the context, meaning the surrounding words. It is believed to work better with small amount of data and with rare words. For skip-gram the one-hot encoded word vector is passed through the hidden layer and then the output layer to predict context words.

For a sentence 'the cat sat on the mat'. For the word 'sat' with a window of size 2, the target words are 'the', 'cat', 'on', and 'the'. Skip-Gram tries to

predict each of these context words given the input word.

2.3 Differences between continuous bag of words and skip-gram

Skip-Gram usually works well with less data and captures more detailed word relationships. For performance, CBOW is faster since it averages over the context words, but Skip-Gram generally has superior quality, especially with infrequent words.

3 Usefull pages and videos

What are Sentiment Analysis - IBM technology

What is Sentiment Analysis? An Introduction - Eye on Tech

How Sentiment Analysis works — AI — NLP - AI Sciences

Word Embedding and Word2Vec, Clearly Explained!!! - StatQuest