Sentiment Analysis

1. Obtaining the IMDb(Internet Movie Database) movie review dataset

Dataset address: <http://ai.stanford.edu/~amaas/data/sentiment/>

The movie review dataset consists of 50,000 polar movie reviews that are labeled as either positive or negative; here ,positive means that a movie was rated with more than six stars on IMDb, and negative means that a movie was rated with fewer than five stars on IMDb

1. **Bag-of-words** model

Bag-of-words model allows us to represent text as numerical feature vectors. The idea behind the bag-of-words model is quite simple and can be summarized as follows:

1. We create a vocabulary of unique tokens-for example, words-form the entire set of documents

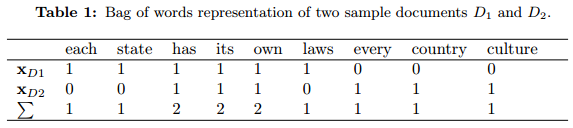
Let D1 and D2 be two documents in a training dataset:

* D1: Each state has its own laws
* D2: Every country has its own culture

The vocabulary could be written as:



1. We construct a feature vector from each document that contains the counts of how often each word occurs in the particular document



The feature vectors will consist of mostly zeros, which is why we call them sparse.

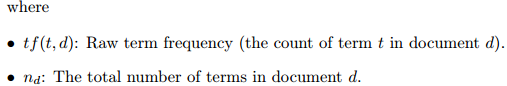
Class: CountVectorizer

Given the example in table 1 one question is whether the 1s and 0s of the feature vectors are binary counts (1 if the word ocuurs in a particular document, 0 otherwise) or absolute counts (how often the word occurs in each document). It depends on which probabilistic model is used. E.g.for Naïve Bayes classifier: The Multinomial or Bernolli model

**Raw term frequencies**: tf(t,d)- the number of times a term t occurs in a document d

In practice , the term frequencies is often normalized by dividing the raw term frequency by the document length:





1. **gram** : each item or token in the vocabulary represents a single word

n- gram: contiguous sequences of items in NLP – words, letters, or symbols – is also called n-gram. The choice of the number n in the n-gram model depends on the particular application. For example, Andelka Zecevic found in his study that *n*-grams with 3 *≤ n ≤* 7 were the best choice to determine authorship of Serbian text documents . In a different study, the *n*-grams of size 4 *≤ n ≤* 8 yielded the highest accuracy in authorship determination of English text books and Kanaris *e. al.* report that *n*-grams of size 3 and 4 yield good performances in anti-spam filtering of e-mail messages.

1. Term frequency—inverse document frequency (tf-idf)

Can be used to downweight those frequently occurring words in the feature vectors. The tf-idf can be defined as the product of the term frenquency and the inverse document frequency:



Tf(t,d) refers to term frequency.

Inverse document frequency idf(t,d) can be calculated as:



Where nd is the total number of documents , and df(d,t) is the number of documents d that contain the term t.Adding the constant 1 to the denominator is optional and serves the purpose of assigning a non-zero value to terms that occur in all training samples; the log is used to ensure that low document frequencies are not given too much weight

Scikit: TfidfTransformer ,CountVectorizer

1. Cleaning text data

clean the text data by stripping it of all unwanted characters.

Use Python’s regular expression (regex) library, re, to accomplish this task.

1. Processing documents into tokens (**Tokenization**)

**Tokennization** describes the general process of breaking down a text corpus into individual elements that serve as input for various natural language processing algorithms. Usually,tokenization is accompanied by other optional processing steps , such as the removal of stop words and punctuation characters, stemming or lemmatizing, and the construction of n-grams.

Typitcal steps: 1. Splits a sentence into individual words, 2. removes punctuation, 3.converts all letters to lowercase

One way to tokenize documents it to split them into individual words by splitting the cleaned document at its whitespace characters;

Another useful technique is word **stemming**, which is the process of transforming a word into its root form that allows us to map related words to the same stem. Original stemming algorithm : Porter stemmer

**Lemmatization:** aims to obtain the canonical (grammatically correct) forms of the words. Lemmatization is computationally more difficult and expensive than stemming, and in practice, both stemming and lemmatization have little impact on the performance of classification.

Scikit : nltk

**Stop-word removal:** stop words are simply those words that are extremely common in all sorts of texts and likely bear no (or only little) useful information that can be used to distinguish between different classes of documents. Examples of stop-words are is, and, has, and the like. Removing stop-words can be useful if we are working with raw or normalized term frequencies rather than tf-idfs, which are already downweighting frequently occurring words.

**Scikit: nltk**

1. **Training a logistic regression model for document classification**

**Code:github.**

A still very popular classifier for text classification is the Naïve Bayes classifier, which gained popularity in applications of e-mail spam filtering. Naïve Bayes classifiers are easy to implement, computationally efficient, and tend to perform particularly well on relatively small dataset compared to other algorithms.

1. **Working with bigger data – online algorithms and out-of-core learning**

Stochastic gradient descent is an optimization algorithm that updates the model’s weights using one sample at a time. In this section, we will make use of the partial\_fit function of the SGDclassifier in scikit-learn to stream to the documents directly from our local drive and train a logistic regression model using small minibatches of documents

The accuracy of the model is slightly below the accuracy that we achieved in the previous section using the grid search for hyperparameter tuning.However,out-of-core learning is very memory-efficient and took less than a minute to complete.

1. Other useful models for text classification:
2. bag of words
3. Latent Dirichlet allocation
4. Word2vec (Efficeient Estimation of Word Representations in Vector space)

Reference:

1. Python machine learning
2. Naïve Bayes and Text Classification Introduction and Theory