Comparison of algorithms classifying text articles, by assigning to them predefined tags

Grzegorz Meller, Mateusz Rock
Data Engineering

1. **Topic**

   Text categorization is the task of assigning a number of appropriate categories to a text document. This categorization process has many applications such as document routing or document management. Traditionally each incoming document is analyzed and categorized manually by domain experts based on the content of the document. Extensive human resources have to be spent on carrying out such a task. To facilitate the process of text categorization, automatic categorization schemes are required. Its goal is to build categorization models that can be used to classify text documents automatically.

   The goal of our project is to implement different classification algorithms and compare results of the outcomes for data from the database. For our project we will use two training datasets. First with articles with few tags assigned, afters successful implementation, we will try to use our algorithms in "extreme classification" where many tags are assigned to each article using Extreme Classification Repository. After all we will compare which algorithms work better and more efficient on simple classification and extreme one.

2. **Process of text pre-processing**

   1. Collect data
   2. Delete characters like '.' '!' ',' ';' etc.
   3. Delete so called stop words like 'a', 'the', 'about' etc.
   4. Delete from words past and present forms e.g. "-ed", "-ing"
   5. Separate words from sentences and count they frequency in each article. It is so called Bag-of-words. (Here also different forms can be applied like vectors of words, it depends on algorithm).
   6. Apply one of the algorithms for classification.

3. **K-nearest neighbors (KNN)**

   The first machine-learning algorithm we'll look at is K-Nearest Neighbors (kNN). It works like this: we have an existing set of example data, our training set. We have labels for all of this data-we know what class each piece of the data should fall into. When we're given a new piece of data without a label, we compare that new piece of data to the existing data, every piece of existing data. We then take the most similar pieces of data (the nearest neighbors) and look at their labels. We look at the top k most similar pieces of data from our known dataset; this is where *k* comes from. (*k* is an integer and it's usually less than 20.) Lastly, we take a majority vote from the k most similar pieces of data, and the majority is the new class we assign to the data we were asked to classify

   Pseudocode of kNN function will look as below. inX is a piece of data we want to classify.

   For every point in our dataset:
   calculate the distance between inX and the current point
   sort the distances in increasing order
   take k items with lowest distances to inX
   find the majority class among these items
   return the majority class as our prediction for the class of inX

Distances are calculated using Euclidean distance where distance between two vectors, xA and xB, with two elements, is given by

$$d = \sqrt{(xA_0 - xB_0)^2 + (xA_1 - xB_1)^2 + ... + (xA_i - xB_i)^2}$$

Where *i* is the number of features. In our example it will be probably the number of all words counted in article, and A and B will be numbers of occurrences of this word in article.

4. Naive Bayes

$$p(c_i|w) = \frac{p(w|c_i)p(c_i)}{p(w)}$$

Variable *w* is vector, that is, we have many values, in our case as many values as words in our vocabulary (all unique words in all of our articles). Variable c_i represents predefined tag (e.g. Computer Science, Biology, Politics etc.). So on the left side of equation we are looking for probability that tag number *i* will be assigned to article, knowing that *w* vector of words exists in the article. We can calculate $p(c_i)$ by adding up how many times we see tag *i* and then dividing by the total number of articles. How can we get $p(w|c_i)$? If we expand *w* into individual features, we could rewrite this as $p(w_0|c_i)p(w_1|c_i)p(w_2|c_i)...p(w_N|c_i)$.

Pseudocode for this function would look like this:

Count the number of articles in each tag
for each training article:
    for each tag:
        if a word appears in an article → increment the count for that word
        increment the count for words
    for each tag:
        for each word:
            divide the word count by the total words count to get conditional probabilities
    return conditional probabilities for each tag

5. Deep Neural Networks

The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. In our example, neural network will be able to classify article, buy learning from entire training dataset, where all articles are labeled with tags. Neural Network in its hidden layer would set appropriate wages (basing on training set), so that it would be able to classify unlabeled article.

6. **Support Vector Machines**

   In our problem, we will also use SVM. SVM is an abstract concept of a machine, the operation of which is similar to the operation of a classifier. Teaching of such a machine is aimed at determining the plane which will separate individual classes with a maximum margin. In our problem, in svm we will probably use a polynomial or nuclear function, because our svm will contain as many classes as words in a given article, on this basis we will try to find a hyper sample that will separate particular groups.

7. **Training Datasets**
   a. Easier dataset with few tags assigned to each article: https://www.kaggle.com/aashita/nyt-comments
   b. Extreme Classification Repository (We will probably use Wiki10-31K): http://manikvarma.org/downloads/XC/XMLRepository.html

8. **References**

   a. Peter Harrington. (2012). Machine Learning in action.
   b. Matthew Kirk. (2017). Thoughtful Machine Learning with Python.
   c. https://www.researchgate.net/publication/257432190_Using_kNN_model_for_automatic_text_categorization
   d. https://appliedmachinelearning.blog/2018/01/18/conventional-approach-to-text-classification-clustering-using-k-nearest-neighbor-k-means-python-implementation/
   e. https://www.cs.cornell.edu/people/tj/publications/joachims_97b.pdf
   f. http://www.kernel-machines.org/
   g. https://en.wikipedia.org/wiki/Artificial_neural_network
   h. https://github.com/thoughtfulml/examples-in-python/tree/master/support_vector_machines