Naïve Bayes for Article Classification

This problem looks at an application of Naive Bayes for multiclass text classification. First, you will use the *New York Times* Developer API to fetch recent articles from several sections of the *Times*. Then, using the simple Bernoulli model for word presence, you will implement a classifier which, given the text of an article from the *New York Times*, predicts the section to which the article belongs.

First, register for a *New York Times* Developer API key and request access to the Article Search API. After reviewing the API documentation, write code to download the 2,000 most recent articles for each of the Arts, Business, Obituaries, Sports, and World sections. (Hint: Use the nytd_section_facet facet to specify article sections.) The developer console may be useful for quickly exploring the API. Your code should save articles from each section to a separate file in a tab-delimited format, where the first column is the article URL, the second is the article title, and the third is the body returned by the API.

You need to share the articles you downloaded with me.

Next, implement code to train a simple Bernoulli Naive Bayes model using these articles. You can consider documents to belong to one of C categories, where the label of the ith document is encoded as $y_i \in 0, 1, 2, \ldots C$—for example, Arts = 0, Business = 1, etc.—and documents are represented by the sparse binary matrix X, where $X_{ij} = 1$ indicates that the ith document contains the jth word in our dictionary.

You train by counting words and documents within classes to estimate θ_{jc} andθ_c:

$$\hat{\theta}_{jc} = \frac{n_{jc} + \alpha - 1}{n_c + \alpha + \beta - 2}$$

$$\hat{\theta}_c = \frac{n_c}{n}$$

where n_{jc} is the number of documents of class c containing the jth word, n_c is

the number of documents of class c, n is the total number of documents, and the user-selected hyperparameters α and β are pseudocounts that "smooth" the parameter estimates. Given these estimates and the words in a document x, you calculate the log-odds for each class (relative to the base class c = 0) by simply adding the class-specific weights of the words that appear to the corresponding bias term:

$$\log\left(\frac{p(y=c|x)}{p(y=0|x)}\right) = \sum_j \hat{w}_{jc} x_j + \hat{w}_{0c}$$

where

$$\hat{w}_{jc} = \log \frac{\hat{\theta}_{jc}\left(1-\hat{\theta}_{j0}\right)}{\hat{\theta}_{j0}\left(1-\hat{\theta}_{jc}\right)}$$

$$\hat{w}_{0c} = \sum_j \log \frac{1-\hat{\theta}_{jc}}{1-\hat{\theta}_{j0}} + \log \frac{\hat{\theta}_c}{\hat{\theta}_0}$$

Your code should read the title and body text for each article, remove unwanted characters (e.g., punctuation), and tokenize the article contents into words, filtering out stop words (given in the stopwords file). The training phase of your code should use these parsed document features to estimate the weights w, taking the hyperparameters α and β as input. The prediction phase should then accept these weights as inputs, along with the features for new examples, and output posterior probabilities for each class.

Evaluate performance on a randomized 50/50 train/test split of the data, including accuracy and runtime. Comment on the effects of changing α and β. Present your results in a (5×5) confusion table showing counts for the actual and predicted sections, where each document is assigned to its most probable section. For each section, report the top 10 most informative words. Also present and comment on the top 10 "most difficult to classify" articles in the test set.

Source: Page 108 of "Doing Data Science" from Cathy O'Neil & Rachel Schutt