Assignment 4 (W4242)

This assignment is due on Wednesday, November 20th, 6:10pm before class. You can discuss with other people, but make sure write down their names on your homework.

Q1[**MapReduce/Feature Generation**] Feature generation is at the core of building good models. Generating good feature can be computationally expensive. Last week we learned that computation can be sped up by decomposing a problem and applying mapreduce. This question will focus on using mapreduce to generate features, specifically features of the Kaggle competition dataset.

(a) If you recall there are two phases in a mapreduce process. A map phase where you divide the data into smaller subproblems and get the results, and a reduce phase where you combine the answers from the subproblems to get the answer you were looking for.

You will need to implement the mapreduce algorithm for computing word count features for the Kaggle dataset. This means that you must segment your data, write a mapper, and write a reducer. You can find out more about mapreduce by looking at the slides Aaron handed out and searching the web for examples.

You can implement this algorithm in R, or if you wish you can use the Google Compute Engine credit and use Hadoop. Either way you will have to implement a separate mapper and reducer. And for either case you will need to submit your solution.

A good tutorial on deploying and using Hadoop on Compute Engine is: `https://github.com/GoogleCloudPlatform/solutions-google-compute-engine-cluster-for-hadoop`. There is also a short screencast video that helps through the process of setting up and using the script: `http://www.youtube.com/watch?v=se9vV8eIZME`.

(b) You must also come up with 20 other features based on the Kaggle dataset. These features can be simple (e.g., the length of a sentence), but they need to be distinct from each other. You do not have to use mapreduce to solve this question.

(c) Use those features to generate a model and submit that model to the Kaggle website.

Q2[Naive Bayes–NYTimes API] 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[1] and request access to the Article Search API.[2] 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[3] 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.

Next, implement code to train a simple Bernoulli naive Bayes model using these articles. We consider documents to belong to one of C categories, where the label of the i-th document is encoded as $y_i \in \{0, 1, 2, \ldots, C-1\}$. For example, Arts= 0, Business= 1, etc. And documents are represented by the sparse binary matrix X, where $X_{ij} = 1$ indicates that the i-th document contains the j-th word in our dictionary. We train by counting words and documents within classes to estimate θ_{jc} and θ_c:

$$\begin{aligned} \hat{\theta}_{jc} &= \frac{n_{jc} + \alpha - 1}{n_c + \alpha + \beta - 2} \\ \hat{\theta}_c &= \frac{n_c}{n} \end{aligned}$$

where n_{jc} is the number of documents of class c containing the j-th 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, we 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:

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

[1] `http://developer.nytimes.com/apps/register`
[2] `http://developer.nytimes.com/docs/read/article_search_api`
[3] `http://prototype.nytimes.com/gst/apitool/index.html`

where

$$\begin{aligned} \hat{w}_{jc} &= \log \frac{\hat{\theta}_{jc}(1-\hat{\theta}_{j0})}{\hat{\theta}_{j0}(1-\hat{\theta}_{jc})} \\ \hat{w}_{0c} &= \sum_j \log \frac{1-\hat{\theta}_{jc}}{1-\hat{\theta}_{j0}} + \log \frac{\hat{\theta}_c}{\hat{\theta}_0} \end{aligned}$$

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 $\hat{w}$, 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-by-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. Briefly discuss how you expect the learned classifier to generalize to other contexts, e.g. articles from other sources or time periods.

(Feel free to use packages to train the simple Bernoulli naive Bayes model. However, coding it up by yourself deserves extra credits.)

Q3[Blog Response] Please read and respond to:

`http://columbiadatascience.com/2013/10/27/data-journalism-redux/`