Data-driven modeling
APAM E4990
Spring 2012

Homework 02

1. Cross-validation for polynomial regression

In this problem you will use cross-validation to determine a best-fit polynomial of the form

$$\hat{y}_K(x; w) = \sum_{k=0}^{K} w_k x^k \tag{1}$$

for the $N = 200$ observations provided in `polyfit.tsv` by minimizing squared loss. As both the degree K and the weights w are unknown, use the closed-form solutions to the normal equations to find the best w for a given K and use cross-validation with a suggested 50%/50% training/testing split to select the best value of K.

Recall from class that under the squared loss

$$\mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \left[y_i - \hat{y}_K(x_i; w)\right]^2 \tag{2}$$

the solution for the best-fit parameters $\hat{w}$ is

$$\hat{w} = \left(\Phi^T \Phi\right)^{-1} \Phi^T y, \tag{3}$$

where y is the $N \times 1$ vector of outputs and Φ is an $N \times (K + 1)$ matrix with elements $\Phi_{ik} = (x_i)^k$, i.e. the k-th power of the i-th input x_i for $k = 0, \ldots, K$. The vector $\hat{w}$ is length $(K+1)\times 1$ and gives estimates of the degree-K polynomial coefficients for the function

Provide a plot of the training and test error as a function of the polynomial degree which indicates the optimal degree. For this optimal degree, also provide a scatter plot of the data with the best-fit model overlayed. Report the coefficients for the best-fit model.

2. Naive Bayes for article categorization

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 discussed in class, 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. Recall from class that we consider documents to belong to one of C categories, where the label of the i-th document is encoded as $y_i \in \{0, 1, 2, \ldots, C\}$—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:

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

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

[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`

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.