Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.008 Introduction to Inference
Fall 2021

Computational Lab IV

Issued: Friday, November 5, 2021 **Due:** Friday, November 19, 2021

Spam Filtering

We are going to build a spam filter. Let $C \in \{\texttt{spam}, \texttt{ham}\}$ be a random variable that represents the type of an email message; non-spam emails are commonly referred to as "ham." We assume that each email is independent of all other emails. Let $\{\texttt{w}_1, \ldots, \texttt{w}_n\}$ be the set of all words used in all emails. Let $Y_i \in \{0, 1\}$ be a random variable that takes on value 1 if the email contains word $\texttt{w}_i$ and 0 if it does not. Our goal is to build a classifier for identifying spam from a training data set of emails that have been labeled as spam or ham.

Code and Data

The file `spam_filtering.zip` contains the functions:

`naivebayes.py`	Your code goes here.
`logistic.py`	Your code goes here.
`util.py`	Python functions/classes that will be useful as you write your classifier. Please do not modify.

and the data[1]

`data/spam/`	Spam emails.
`data/ham/`	Non-spam (i.e., ham) emails.
`data/testing/`	Test data for you to classify.

Part I: Naïve Bayes Classifier

In building this classifier, we model a word's presence in the email as independent of all other words, given the type of email. Specifically, our (generative) model is

$$p_{C,Y_1,\ldots,Y_n}(c, y_1, \ldots, y_n; \boldsymbol{\theta}) = p_C(c; s) \prod_{i=1}^{n} p_{Y_i|C}(y_i|c; p_i, q_i),$$

with

$$p_C(c; s) = \begin{cases} s & c = \texttt{spam} \\ 1 - s & c = \texttt{ham}, \end{cases}$$

[1] The data used in this assignment comes from a preprocessed version of the Enron email database. See V. Metsis, I. Androutsopoulos and G. Paliouras, "Spam Filtering with Naïve Bayes—Which Naïve Bayes?" in *Proc. Conf. Email and Anti-Spam (CEAS-2006)*, (Mountain View, CA), 2006.

and, for $i = 1, \ldots, n$,

$$p_{Y_i}(y_i|c; p_i, q_i) = \begin{cases} q_i^{y_i}(1-q_i)^{1-y_i} & y_i \in \{0,1\},\ c = \texttt{spam} \\ p_i^{y_i}(1-p_i)^{1-y_i} & y_i \in \{0,1\},\ c = \texttt{ham}, \end{cases}$$

where $\boldsymbol{\theta} = (s, p_1, \ldots, p_n, q_1, \ldots, p_n)$ are the model parameters (all with values between 0 and 1) to be learned from the training set. In this part of the lab, we only work with functions in `naivebayes.py`.

(a) *Training.* First, we estimate the parameters of the model.

i) Determine the maximum likelihood (ML) estimates of the parameters $(s, p_1, \ldots, p_n, q_1, \ldots, p_n)$, based on a training set consisting of k emails. This part has no coding.

ii) Implement the function `get_counts` that counts the number of files each word occurs in.

iii) Implement the function `get_log_probabilities` that computes the log of a smoothed frequency for each word. See Recitation 12 notes for more details on Laplace smoothing.

iv) Implement the function `learn_distributions`. This function takes two lists of filenames, one for spam and one for ham, and computes ML estimates of parameters $(s, p_1, \ldots, p_n, q_1, \ldots, p_n)$.

(b) *Testing.* Implement the function `classify_message`, which computes the MAP estimate of the type of an email from the test dataset. The function should use Bayes' rule with the ML parameter estimates you obtained in the training. You can test your code by running

```
python naivebayes.py data/testing/ data/spam/ data/ham/
```

How well does your classifier perform?

(c) Estimating the spam frequency s from data depends heavily on the number of examples in your training set. In the real world, it is often difficult to find good training examples for ham, since nobody wants to give out their private email for the world to read. As a result, spam datasets often have many more spam examples than ham examples. By setting s to reflect your belief of how often you get spam emails, we can adjust how much the algorithm favors catching spam at the expense of falsely flagging a ham message. Try setting s to a few values, and briefly explain what happens to your performance as s increases and decreases.

Part II: Logistic Regression Classifier

We next use a logistic model for posterior distribution of email type C, with the simple features $g_i(y_i) = y_i$ for $i = 1, \ldots, n$; specifically, our (discriminative) model is

$$p_{C|Y_1,\ldots,Y_n}(c|y_1,\ldots,y_n;\boldsymbol{\theta}) = \begin{cases} \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i y_i\right) & c = \texttt{spam} \\ 1 - \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i y_i\right) & c = \texttt{ham}, \end{cases}$$

with

$$\sigma(u) = \frac{1}{1 + \mathrm{e}^{-u}}$$

denoting the sigmoid function defined in class, and where $\boldsymbol{\theta} = (\theta_0, \ldots \theta_n)$ are the model parameters to be learned from the training set.

Given training data in the form of k i.i.d. labeled data examples

$$\left(c, y_{1,\ldots,n}\right)^{(1,\ldots,k)} = \left(c, y_{1,\ldots,n}\right)^{(1)}, \ldots, \left(c, y_{1,\ldots,n}\right)^{(k)},$$

we learn the parameters of the model above by maximizing the component of the (log) likelihood used in discriminative modeling, or alternatively, by minimizing the negative log-likelihood, viz.,

$$\begin{aligned} \varphi\left(\boldsymbol{\theta}; \left(c, y_{1,\ldots,n}\right)^{(1,\ldots,k)}\right) &= -\sum_{j=1}^{k} \ln p_{C|Y_1,\ldots,Y_n}(c^{(j)}|y_1^{(j)},\ldots,y_n^{(j)};\boldsymbol{\theta}) \\ &= -\sum_{j=1}^{k}\Bigg[\mathbb{1}(c^{(j)} = \texttt{spam}) \ln \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i y_i^{(j)}\right) \\ &\qquad + \mathbb{1}(c^{(j)} = \texttt{ham}) \ln\left(1 - \sigma\left(\theta_0 + \sum_{i=1}^{n} \theta_i y_i^{(j)}\right)\right)\Bigg], \end{aligned} \tag{1}$$

where to obtain the last equality we have used that a $\mathtt{B}(p)$ distribution over $\mathcal{Z} = \{\mathtt{a}_1, \mathtt{a}_2\}$ with $p_Z(\mathtt{a}_1) = p$ can be expressed in the convenient form

$$p_Z(z) = p^{\mathbb{1}(z=\mathtt{a}_1)}(1-p)^{\mathbb{1}(z=\mathtt{a}_2)}.$$

We carry out this minimization numerically through a procedure referred to as *gradient descent*. Briefly, the procedure evaluates the gradient of the function $\varphi(\cdot)$ with respect to parameters θ and takes a step towards lower value of the function along its gradient. When the value of the function does not increase with subsequent steps, we have arrived at a (local) maximum. We have provided an implementation of the gradient descent for you to use. In this part of the lab, we only work with functions in `logistic.py`.

(d) *Training.* We first estimate the model parameters.

i) Implement the function `extract_features` that extracts the binary features $y_1, \ldots, y_n$ from the input email message as defined at the start of the problem.

ii) Implement the function `logistic_eval` that computes

$$\sum_{j=1}^{k} \ln p_{C|Y_1,\ldots,Y_n}(c^{(j)}|y_1^{(j)}, \ldots, y_n^{(j)}; \boldsymbol{\theta}).$$

Note that the cost function $\varphi(\cdot)$ in (1) is just the negative of this quantity.

iii) Obtain an expression for the derivative of the cost function

$$\varphi\left(\boldsymbol{\theta}; \left(c, y_{1,\ldots,n}\right)^{(1,\ldots,k)}\right) \tag{2}$$

with respect to the each of the constituent parameters $\theta_0, \ldots, \theta_n$, including its derivation in your writeup. Then implement the function `logistic_derivative` that computes this derivative.

Hint: You can take the derivative for one data point at a time, and then sum the derivatives for all data points. Also, you may find it useful to use the following properties of the sigmoid function from class:

$$\sigma(-u) = 1 - \sigma(u), \qquad \text{and} \qquad \frac{\mathrm{d}}{\mathrm{d}u}\sigma(u) = \sigma(u)\,\sigma(-u).$$

iv) Implement the function `train_logistic`. This function takes two lists of filenames, one for spam and one for ham, and computes the ML estimates of parameters $\theta_0, \ldots, \theta_n$. Set the `SHOW_LOSS_PLOT` global variable at the top of `logistic.py` to `True` to observe the behavior of the cost function (2) over iterations of the gradient descent procedure. Include the plot in your writeup.

(e) *Testing.* Implement the function `classify_message` in `logistic.py`, which computes the MAP estimate of the email's type with the ML parameter estimates you obtained in training. You can test your code by running

```
python logistic.py data/testing/ data/spam/ data/ham/
```

How well does your classifier perform? How well does it perform compared to your previous classifier from `naivebayes.py`?

(Note that we've set `logistic.py` to use only 100 examples from each of spam and ham for training, governed by the `NUM_EXAMPLES` global variable at the top of `logistic.py`. It's very slow to train on the full data, and the results aren't much different, so it's fine to train your model on just this smaller set.)

For the writeup, report the performance for `NUM_EXAMPLES`=100.

(Bonus) Note that the performance of your classifier also depends on the `learning_rate` parameter for the training (defined in `optimize_theta` method). Feel free to experiment with different values. How are the training time and convergence of parameters be affected? For the writeup, please use the given `learning_rate` of 0.05 .

(f) Implement an alternative function `extract_features` with a different choice of features. For example, you could choose to use word *frequencies* instead of just binary labels for whether a word appears in the file, as we did previously. Describe your implementation of `extract_features`, and compare the results with those from the previous implementation. What do you think accounts for the difference in performance, if any?

What to hand in: Upload the completed files `naivebayes.py` and `logistic.py` to Gradescope along with a PDF file with your writeup. The writeup should include your answers for all parts. Scans of handwritten work are fine but, as always, please make sure it is legible; we cannot grade what we cannot decipher. The Gradescope autograder has a visible test case, please make sure your code runs and passes the test.