CMPS242 Homework #3 – Logistic Regression

Benjamin Sherman

&

Zayd Hammoudeh

October 20, 2017

1 Homework Objective

Develop a logistic regression-based learner that can identify SMS text messages as either SPAM or ham (i.e., not SPAM). Additional requirements include:

- Learner should implement a batch gradient descent
- Create a Jupyter (i.e., IPython) notebook to document all work

Four extra credit tasks were also possible for this homework. They are:

1. Support a bias term in the weight vector ($\mathbf{w}$) that is not regularized
2. Experiment with different regularizers, such as $\lambda \|\mathbf{w}\|_1$.
3. Implement the stochastic gradient descent algorithm
4. Implement the exponentiated gradient algorithm ($\text{EG}^{\pm}$)

2 Gradient Descent Update Rule

Logistic regression is a subcategory of binary classification. Define the logistic (sigmoid) function as shown in Eq. (1).

$$\sigma(a) = \frac{e^a}{1 + e^a} \tag{1}$$

As shown in Figure 1, this function is bounded between 0 and 1 making it ideal for binary classification since it can be used to represent probabilities without additional normalization. The following notes show the derivation of the logistic update rule.[1]

Define Z as a random variable that maps one classification value (e.g., ham) to 0 and the other classification result (e.g., SPAM) to 1. We could then define the probability of a 0 or a 1 as shown in Eq. (2) and (3) respectively. This can be written compactly as shown in Eq. (4).

$$\Pr\{Z = 0|a\} = 1 - \sigma(a) \tag{2}$$
$$\Pr\{Z = 1|a\} = \sigma(a) \tag{3}$$
$$\Pr\{Z = z|a\} = \sigma(a)^z(1 - \sigma(a))^{1-z} \tag{4}$$

Given a target classification vector $\mathbf{y}$ and the relation $a =$, the likelihood (L) is in turn defined as shown in Eq. (**??**). Since each element is independent, this probability can be transformed into a product as shown in Eq. (6), where $n = |\mathbf{y}|$.

$$L(\mathbf{w}) = \Pr\{\mathbf{y}|\mathbf{w}^{\mathrm{T}}\mathbf{x}\} \tag{5}$$
$$= \prod_{i=1}^{n} \Pr\{y_i|\mathbf{w}^{\mathrm{T}}\mathbf{x}_i\} \tag{6}$$

As with homework #1, the value that maximizes the likelihood of a non-negative function also maximizes the log likelihood. Hence, Eq. (6) can be transformed to Eq. (11).

[1] This proof is based on the lecture notes of Andrew Ng.

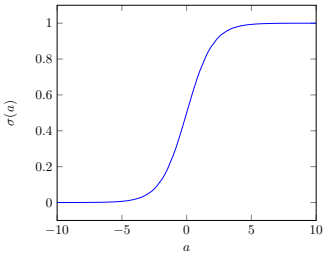

Figure 1: Plot of the Logistic Function

$$\ln L(\mathbf{w}) = \ln\left(\prod_{i=1}^{n} \Pr\{y_i|\mathbf{w}^{\mathrm{T}}\mathbf{x}_i\}\right) \tag{7}$$

$$= \sum_{i=1}^{n} \ln \Pr\{y_i|\mathbf{w}^{\mathrm{T}}\mathbf{x}_i\} \tag{8}$$

$$= \sum_{i=1}^{n} \ln\left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)^{y_i}(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))^{1-y_i}\right) \tag{9}$$

$$= \sum_{i=1}^{n} \ln\left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)^{y_i} + \ln\left((1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))^{1-y_i}\right)\right) \tag{10}$$

$$= \sum_{i=1}^{n} y_i \ln \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i) + (1-y_i)\ln(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) \tag{11}$$

To find the maximizing likelihood, the derivative is taken and set equal to 0. We use the identity $\frac{\partial\sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$ without proof, but the proof is trivial. Hence the maximizing value is is shown in Eq. (15); note that the *chain rule* was used for this derivative.

$$0 = \sum_{i=1}^{n}\left(y_i\frac{1}{\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}\frac{\partial\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{\partial\mathbf{w}} - (1-y_i)\frac{1}{1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}\frac{\partial\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}{\partial\mathbf{w}}\right) \tag{12}$$

$$= \sum_{i=1}^{n}\left(y_i\frac{1}{\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))\mathbf{x}_i - (1-y_i)\frac{1}{1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)}\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))\mathbf{x}_i\right) \tag{13}$$

$$= \sum_{i=1}^{n}\left(y_i(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)) - (1-y_i)\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)\right)\mathbf{x}_i \tag{14}$$

$$= \sum_{i=1}^{n}(y_i - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i))\mathbf{x}_i \tag{15}$$

Eq. (15) represents the maximizing gradient for the log likelihood. This can then be substituted into the update rule for stochastic gradient descent, where η is the hyperparameter, learning rate.

$$\mathbf{w}_{t+1} := \mathbf{w}_t + \eta\nabla \ln L(\mathbf{w}) \tag{16}$$

We can then substitute into Eq. (16), using tensor notation since this is batch update.

$$\mathbf{w}_{t+1} := \mathbf{w}_t + \eta\mathbf{X}^{\mathrm{T}}(\mathbf{y}-\hat{\mathbf{y}})^{\mathrm{T}} \tag{17}$$

Note that that $\mathbf{w}_t$ was used in the gradient since it is not known how to maximize on w_{t+1}. In addition, the learning rate, η uses a constant configured by the user an aging term as shown in Eq. (18).

$$\eta = \frac{\eta_0}{t^{0.9}} \tag{18}$$

Our implementation uses the update rule and learning rate in Eq. (17) and (18) respectively for all batch gradient descent experiments in this paper.

3 Text Preprocessor

Before learning can be attempted the natural language text in the form of SMS messages much be turned into a feature vectors. We accomplish this through a multistep process described below (in order).

- *Import CSV File* – Pandas' native CSV support as part of the `DataFrame` object was used.
- *Invalid Character Removal* – The original text files (before my team cleaned them for the class) had invalid, unsupported characters. We cleaned the
- *Punctuation Removal* – The punctuation set is defined by Python's `string` package (i.e., "string.punctuation"). Punctuation removal entailed replacing all punctation symbols with the empty string (e.g., "couldn't" became "couldnt").
- *Capitalization Correction* – Standardizes case so that words with different capitalization are not treated differently.
- *Stop Words Removal* – The set of "stop words" is non-standard. For our experiments, we used the English "`corpus`" dictionary bundled with the Natural Language Toolkit (NLTK).

4 Jupyter Notebook Implementation

Jupyter notebooks serve as a intermediary layer that insulates users from the underlying code implementation. We included multiple features into our Jupyter notebook to improve the user experience and provide external learner configuration and data visualization.

4.1 Specifying Hyperparameters

Since Jupyter notebook masks the code from the user, some may find it difficult to experiment with different configurations. To address this limitation, we added a series of widgets to our notebook. They leverage the `ipywidgets` library. The specific hyperparameters that are controllable by widgets are:

- Cross-validation fold count (k)
- Learning algorithm (e.g., standard gradient descent, exponentiated gradient descent, and stochastic gradient descent)
- Regularizer selection (e.g., L_2 or L_1 norm)
- Error calculation formula (e.g., $1 - \textit{Accuracy}$ or root-mean square)
- Maximum number of training epochs
- Learning rate (α)
- λ range – These are the powers of 2 to test.

Figure 2 shows a screenshot from our Jupyter notebook with the widgets included.

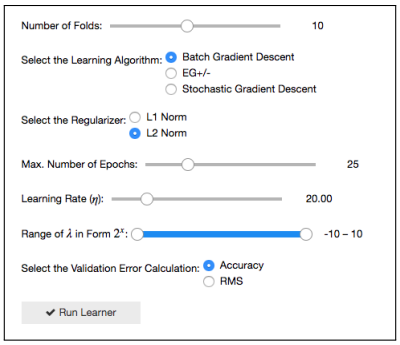

Figure 2: Jupyter notebook experiment configuration widgets

```
from run_learner import run_hw03
train_err, validation_err, test_err = run_hw03(train_data, test_data)

Starting Learner.
Progress: |████████████████████----------------------------| 43.4%
```

Figure 3: Jupyter notebook learner progress indicator

4.2 Learner Progress Indicator

As with most iterative learning algorithms, our learner does provide instantaneous results. Depending on the user specified configuration described in the previous section, it may take several minutes before a run of the algorithm completes. Our implementation provides visual feedback on the learners progress via a text-based progress bar, which is shown in Figure 3.[2]

4.3 Embedded Results Table

To improve the Jupyter notebook experience, we built in support for a table that will display the latest execution's results. The minimum training, validation and test errors are bolded and highlighted in yellow as shown in Figure 4.

4.4 Embedded Result Graphing

We also implemented graphing into our Jupter notebook. This uses Python's `matplotlib` package. An example graph is shown in Figure 5.

5 Experiment Setup

Our solver is implemented in Python 2.7.13. It requires a set of external packages, including:

λ	0	2^{-10}	2^{-9}	2^{-8}	2^{-7}	2^{-6}	2^{-5}	2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}
Training	0.0053	0.0052	**0.0051**	0.0055	0.0077	0.0113	0.0199	0.0433	0.0495	0.0906	0.1390	0.1390
Validation	0.0300	0.0300	**0.0297**	**0.0297**	0.0313	0.0410	0.0590	0.0700	0.0727	0.1067	0.1390	0.1390
Test	0.0233	**0.0229**	0.0241	0.0257	0.0276	0.0385	0.0525	0.0614	0.0634	0.0956	0.1283	0.1283

Figure 4: Example Jupyter notebook table of experimental results

[2]The underlying implementation used for this feature was from an online resource and not developed natively by our team.

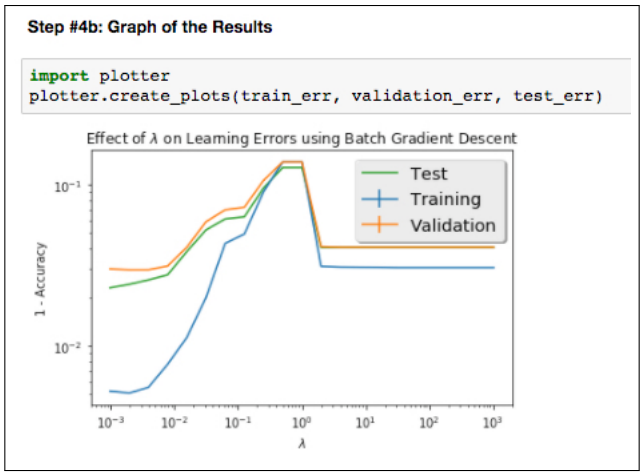

Figure 5: Example Jupyter notebook graph of experimental results

- Pandas
- Scikit-Learn
- NumPy
- IPython
- Matplotlib
- ipywidgets.

6 Extra Credit #1: Bias Term Support

7 Extra Credit #2: Multiple Regularizer Experimentation

8 Extra Credit #3: Stochastic Gradient Descent

In lecture on Thursday October 19, Prof. Warmuth mentioned that there would be extra credit if we implemented stochastic gradient descent. As such, we implemented this

9 Extra Credit #4: Exponentiated Gradient Descent