Homework 2

Evan Sidrow

CSCI 5622

1 Logistic Regression

1.1 Programming questions

See logreg.py (attached to this submission).

1.2 Analysis

1.2.1 *What is the role of the learning rate (eta) on the efficiency of convergence during training?*

A value of η that was too large (1, 0.1) resulted in the loss function oscillating around too much without ever converging, because the gradient descent step always over-shoots the optimal weight vector. This makes the accuracy erratic. However, if η is too small (0.001), then the loss function decreases smoothly, but it never converges to the optimal value, so the accuracy hangs a bit low. However, when η is just right (0.01), the loss function converges somewhat smoothly to an optimal value, and the accuracy is reliably the best it can be. See Figures 1 and 2 for plots of this phenomenon.

1.2.2 *What is the role of the number of epochs on the test accuracy?*

For computational efficiency, I used only 1000 training instances to test the accuracy rate of up to 50 epochs. It is always the case that more epochs corresponds to a better expected test accuracy rate. However, depending upon the value of η, the rate at which the test accuracy gets better changes. Higher values of η mean that it doesn't take many epochs for test accuracy to get high, but then the test accuracy jumps around quite a bit for a while before settling down. Lower values for η mean that it takes more epochs for test accuracy to get high, but it does so smoothly. See Figures 3 and 4 for plots of this phenomenon.

2 Feature Engineering

2.1 Programming questions

See feature_eng.py (attached to this submission).

2.2 Analysis

2.2.1 *What custom features did you add/try (other than n-grams)? How did those addition features affect the model performance? Why do you think those addition features helped/hurt the model performance?*

Before adding any features in addition to the length of the review to the model, the test accuracy was at approximately 50%, which is essentially just guessing.

I tried to add the following features, and the resulting test accuracy of just that feature and the length of the review are as follows:

- Average length of a word in the review (50.3%)
- Average Sentence length (49.9%)

- Percentage of characters that were exclamation points (50.6%)
- Percentage of characters that were question marks (51.2%)
- Percentage of characters that were commas (50.7%)
- Number of instances of "ed " and " was " (indicating past tense) normalized by the length of the review (50.2%)
- Number of instances of "ing " and " is " (indicating present tense) normalized by the length of the review (50.3%)

Clearly, none of these features were very effective in raising accuracy, and I would argue that none of these increases were statistically significant. However, when I included all of these items, I was able to get an accuracy of 52%, which may be more significant, but it still isn't much at all. I think that these features didn't help the model at all because there is not much information in any of the features that I looked at. The average length of a word orsentence has more to do with the vocabulary of the writer than the writer's sentiment, and the types of punctuation has more to do with writing style and strength of feeling than the sentiment itself. In addition, our data set is very noisy and we are only getting one dimensional data from each feature here. One feature that would increase accuracy is the number of instances of "positive" and "negative" words. This is because the actual sentiment of the writer is best expressed in the literal words that are being written about the movie. However, I didn't implement the feature because it would be redundant after putting in n-grams.

2.2.2 *What are unigrams, bigrams, and n-grams? When you added those features to the Feature-Union, what happened to the model performance?*

An n-gram is a sequence of n consecutive words in a passage. for example, the 3-grams of "Hello my name is Evan" are ["Hello my name","my name is","name is Evan"]. Unigrams are equivalent to 1-grams (single words), and bigrams are just 2-grams (two words). Once I added n-grams to the model, the model performance jumped significantly, from 50% to 80%. This is because n-grams get to the actual words that are being written, which contain lots of information about the author's sentiment. Interestingly, the training accuracy went to 100%, which indicates that there is a chance that the training data was being over-fit. To help remedy this, I set the parameters such that a word would need to be seen at least 5 times to be recorded. This dropped the training accuracy to about 99%, but raised the test accuracy by about 1%. This result could just be noise, however.

3 Gradient Descent Learning Rule for Multi-class Logistic Regression

We have that each label is an independent random variable such that:

$$p(Y=c|\boldsymbol{X}) = \frac{\exp(\beta_c^T \boldsymbol{X})}{\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{X})}$$

$$\mathcal{L} = -\log(\prod_{i=1}^{n} \frac{\exp(\beta_{y^{(i)}}^T \boldsymbol{x}^{(i)})}{\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})})$$

$$\mathcal{L} = -\sum_{i=1}^{n} \log(\frac{\exp(\beta_{y^{(i)}}^T \boldsymbol{x}^{(i)})}{\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})})$$

$$\mathcal{L} = \sum_{i=1}^{n} \left(\log(\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})) - \beta_{y^{(i)}}^T \boldsymbol{x}^{(i)} \right)$$

Finding $\nabla_{\beta_{c,j}} \mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \beta_{c,j}} = \frac{\partial}{\partial \beta_{c,j}} \left(\sum_{i=1}^{n} \left(\log(\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})) - \beta_{y^{(i)}}^T \boldsymbol{x}^{(i)} \right) \right)$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{c,j}} = \sum_{i=1}^{n} \frac{\partial}{\partial \beta_{c,j}} \left(\log(\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})) - \beta_{y^{(i)}}^T \boldsymbol{x}^{(i)} \right)$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{c,j}} = \sum_{i=1}^{n} \frac{\partial}{\partial \beta_{c,j}} \left(\log(\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})) \right) - \mathbb{1}\{y^{(i)} = c\} x_j^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{c,j}} = \sum_{i=1}^{n} \frac{\frac{\partial}{\partial \beta_{c,j}} \left(\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)}) \right)}{\sum_{c'=1}^{C} \exp(\beta_{c'}^T \boldsymbol{x}^{(i)})} - \mathbb{1}\{y^{(i)} = c\} x_j^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_{c,j}} = \sum_{i=1}^{n} \frac{x_j^{(i)} \exp(\beta_c^T \boldsymbol{x}^{(i)})}{\sum_{c'=1}^{C} \exp(\beta_c^T \boldsymbol{x}^{(i)})} - \mathbb{1}\{y^{(i)} = c\} x_j^{(i)}$$

$$\nabla_{\beta_{c,j}} \mathcal{L} = \sum_{i=1}^{n} x_j^{(i)} \left(p(Y^{(i)} = c | \boldsymbol{x}^{(i)}) - \mathbb{1}\{y^{(i)} = c\} \right)$$

Which is what we were trying to find.

4 Figures

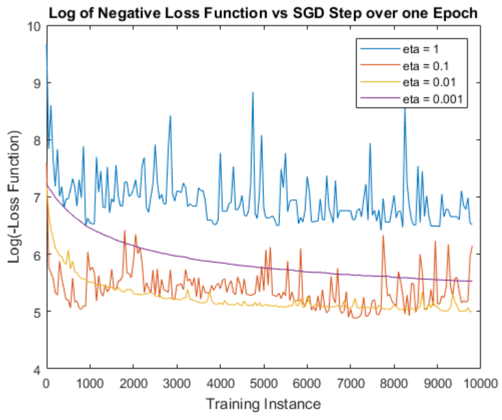

Figure 1: Log of the negative loss function versus number of iterations of stochastic gradient descent. Note that lower values are better fits.

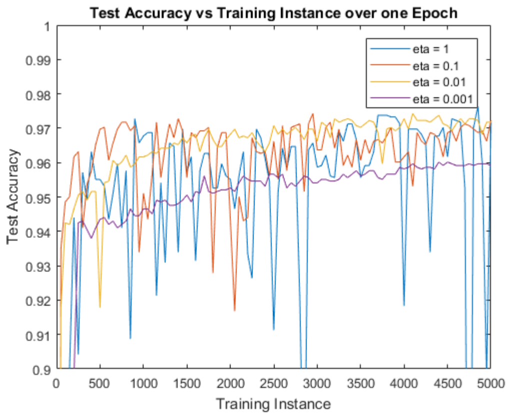

Figure 2: Test accuracy versus number of iterations of stochastic gradient descent.

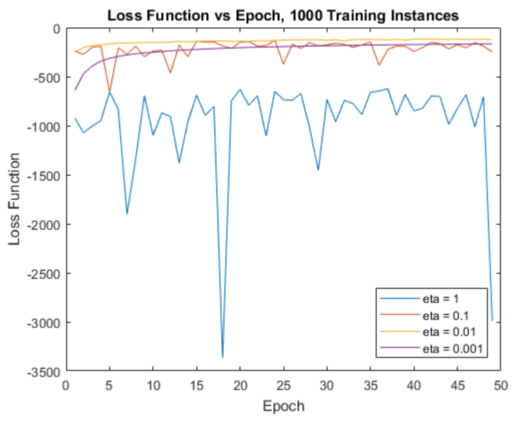

Figure 3: Negative loss function versus number of epochs when 1000 training instances were used. Note that larger values for the loss function are better fits.

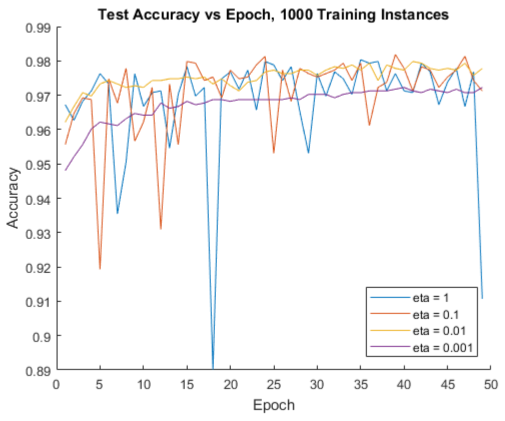

Figure 4: Test accuracy versus number of epochs when 1000 training instances were used.