Machine Learning	**Homework 2**
CSCI 5622 Fall 2017	**Due Time Sep 29, 2017**
Name: Luke Meszar	**CU identitykey: lume0018**

1 Logistic Regression (40pts)

Solution.

1. What is the role of the learning rate (η) on the efficiency of convergence during training?

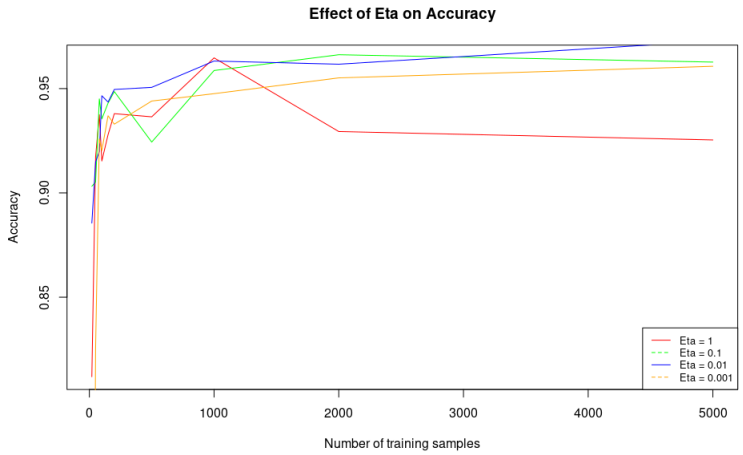

Figure 1: Testing how η affects accuracy

For small η, it can take a long time to converge to the minimum. This can be seen by looking at the graph for $\eta = 0.001$. Over 5000 samples, it never reaches the same level of accuracy that $\eta = 0.01$ or ever $\eta = 0.1$ do. However the graph for $\eta = 0.001$ is the smoothest of the four which means that it follows the gradient of the curve the closest. With a large η, aka 1, it fails to find the minimum which can be seen by the downward slope at the end of the graph. The accuracy for $\eta = 1$ also jumps a lot at the beginning of the graph showing that it overshooting and then undershooting the minimum. From this graph, it appears that $\eta = 0.01$ is the best choice since it converges the fastest. However, $\eta = 0.1$ would be a better choice if there were fewer samples, say 2200 since the model is more accurate there than for $\eta = 0.01$.

2. What is the role of the number of epochs on test accuracy?

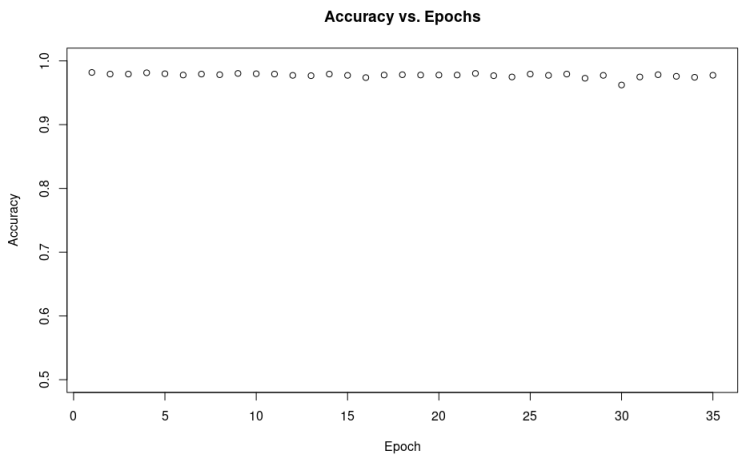

Figure 2: Testing how the number of epochs affects accuracy with $\eta = 0.1$

From fig. 2 it is clear that the number of epochs has almost no effect on accuracy. There is some noise in the data, but for the most part, the accuracy is constant with respect to the number of epochs. This makes sense considering the size of an epoch is 9830. From fig. 1 we see that the accuracy at $\eta = 0.1$ *evens out at 5000. Thus, the model is already been trained as well as it can before even finishing one epoch so increasing the number of epochs doesn't affect anything. The reason you may want to use more epochs is it removes some random variations that are possible if the model is only trained once.*

2 Feature Engineering (40 points)

Solution.

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

   I added four additional features which were tf-idf, most common suffixes, positive words, and negative words. The tf-idf feature was implemented using scikit-learn's TfidfVectorizer function. This feature was the most successful causing the accuracy to raise to 82.5%. This feature counts the number of times a word appears in a review and weights it against the frequency of that word occurring in all reviews. Essentially, weights common words low since they do not provide very much information. This is a good measure of which words are important for classification since tf-idf tries to find meaningful words.

   The next feature I tried was looking at the most common suffixes. Given a list of common suffixes, the percentage of words in a review that ended in a common suffix were calculated. This was the least useful feature only increasing the accuracy of the model by 7%. This is not surprising since it does not seem like a very useful feature.

Since the most common suffixes were used, a lot of words will end in them regardless of any sentimental value they may have. Thus, this features measures nothing about the likelihood of a review being positive or negative.

The final two features I added were similar; calculating the percentage of positive and negative words in a review. Both features increased the accuracy of the model. Due to their similarity, I will only discuss the negative words feature. For each review, the percentage of words that matched one of the negative words in a list was calculated. This is seems like it should be a useful feature since negative reviews are more likely to have negative words. Interestingly, changing the size of the list of negative words affected the accuracy. In a list with 43 word, the accuracy was around 62% but with a list of only 14 words, the accuracy increases to 66%. Similar differences were found with the positive words feature.

2. What are unigrams, bigrams, and n-grams? When you added those features to the FeatureUnion, what happened to the model performance? Why do these features help/hurt?

   An n-gram splits a piece of text into groups of n consecutive words. A unigram splits text into sets of one consecutive word, or what would more commonly be called a bag of words. A bigram will split the text into pairs of consective words. When these feature were added, the accuracy on the test data improved from around 50% at the baseline to 80%. Adding n-gram features to our model improved performance since reviews in common should have similar word structures. For instance, positive reviews are likely to have words like "good", "great", "excellent," and "wonderful." Then, the model is trying to predict on a test review, if the review contains words like that, then it is will predict that the review is positive. Adding bigrams and longer n-grams gives more power to this method until it hits diminishing returns. For instance, adding bigrams allows the model to see phrases like "not good" which may look like an indicator of positive review if there were only unigrams. However, if n gets too large, then the n-grams will be too specific to the review and will lead to overfitting the data.

3 Gradient Descent Learning Rule for Multi-class Logistic Regression (20 pts)

Solution.

1. Derive the negative log likelihood for multi-class logistic regression.

   Let

$$l(\beta) = \prod_{i=1}^{N} \prod_{c=1}^{C} p\left(y_i = c \mid x_i; \beta\right)$$

be the likelihood function. Then, the negative log likelihood function is

$$
\begin{aligned}
-\log l(\beta) &= -\left(\sum_{i=1}^{N}\sum_{c=1}^{C}\log p\left(y_i = c \mid x_i; \beta\right)\right) \\
&= \left[-\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbb{I}(y_i = c)\log\frac{\exp(\beta_c^T x_i)}{\sum_{c'=1}^{C}\exp(\beta_{c'}^T x_i)}\right]
\end{aligned}
$$

where $\mathbb{I}$ is an indicator function that is 1 when $y_i = c$ and 0 otherwise. Then, using properties of logarithms, we get

$$
\mathscr{L} = -\sum_{i=1}^{N}\left[\left(\sum_{c=1}^{C}\mathbb{I}(y_i = c)\beta_c^T x_i\right) - \log\left(\sum_{c'=1}^{C}\exp(\beta_{c'}^T x_i)\right)\right]
$$

The term $\left(\sum_{c=1}^{C}\mathbb{I}(y_i = c)\beta_c^T x_i\right)$ can be simplified to $\beta_{y_i}^T x_i$ since only one of the terms will be nonzero when $y_i = c$. Thus, the simplified log-likelihood function is

$$
-\sum_{i=1}^{N}\left[\left(\beta_{y_i}^T x_i\right) - \log\left(\sum_{c'=1}^{C}\exp(\beta_{c'}^T x_i)\right)\right]
$$

2. The gradient descent learning rule for optimizing weight vectors generalizes to the following form: $\beta_j^{t+1} = \beta_j^t - \eta\nabla\beta_j^t$ where η is the learning rate. Find the $\nabla\beta_{c,j}$ (the parameter for feature x_j in class c) for a multi-class logistic regression model.

   To compute $\nabla\beta_{c,j}$ we compute $\frac{\partial\mathscr{L}}{\partial\beta_{c,j}}$. Then

$$
\begin{aligned}
&\frac{\partial}{\partial\beta_{c,j}}\left[-\sum_{i=1}^{N}\left(\left(\beta_{y_i}^T x_{i,j}\right) - \log\left(\sum_{c'=1}^{C}\beta_{c',j}x_{i,j}\right)\right)\right] \\
&= -\sum_{i=1}^{N}\frac{\partial}{\partial\beta_{c,j}}\beta_{y_i}^T x_i + \frac{\partial}{\partial\beta_{c,j}}\log\left(\sum_{c'=1}^{C}\exp(\beta_{c'}^T x_{i,j})\right) \\
&= -x_{i,j} + \frac{x_{i,j}\exp(\beta_{c,j}x_{i,j})}{\sum_{c'=1}^{C}\beta_{c',j}x_{i,j}} \\
&= x_{i,j}\left[-1 + p(y = c \mid x)\right].
\end{aligned}
$$

   And thus we have computed $\nabla\beta_{c,j}$.