Logistic Regression with L2 regularization

The goal of this assignment is to implement your own logistic regression classifier with L2 regularization. You will do the following:

- Extract features from Amazon product reviews.
- Convert an dataframe into a NumPy array.
- Write a function to compute the derivative of log likelihood function with an L2 penalty with respect to a single coefficient.
- Implement gradient ascent with an L2 penalty.
- Empirically explore how the L2 penalty can ameliorate overfitting.

If you are doing the assignment with IPython Notebook

An IPython Notebook has been provided below to you for this assignment. This notebook contains the instructions, quiz questions and partially-completed code for you to use as well as some cells to test your code.

Make sure that you are using GraphLab Create 1.8.3. See this post for installing the correct version of GraphLab Create.

What you need to download

If you are using GraphLab Create:

- Download the Amazon product review dataset (subset) in SFrame format. Notice the subset suffix: amazon_baby_subset.gl.zip
- Download the companion IPython notebook: module-4-linear-classifier-regularization-assignment-blank.ipynb
- Download the list of 193 significant words: important_words.json
- If you are using Amazon EC2, download the binary files for NumPy arrays: module-4-assignment-numpy-arrays.npz. See the companion notebook for the instructions.
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

If you are not using GraphLab Create:

- If you are using SFrame, download the Amazon product review dataset (subset) in SFrame format: amazon_baby_subset.gl.zip
- If you are using a different package, download the Amazon product review dataset (subset) in CSV format: amazon_baby_subset.csv.zip
- Download the list of 193 significant words: important_words.json

If you are using GraphLab Create and the companion IPython Notebook

Open the companion IPython notebook and follow the instructions in the notebook.

If you are using other tools

This section is designed for people using tools other than GraphLab Create. **You will not need any machine learning packages** since we will be implementing logistic regression from scratch. **We highly suggest you use SFrame since it is open source.** In this part of the assignment, we describe general instructions, however we will tailor the instructions for SFrame.

- If you choose to use SFrame, you should be able to follow the instructions in the next section and complete the assessment. **All code samples given here will be applicable to SFrame**.
- You are free to experiment with any tool of your choice, but **some many not produce correct numbers for the quiz questions.**

If you are using SFrame

Make sure to download the companion IPython notebook: module-4-linear-classifier-regularization-assignment-blank.ipynb. You will be able to follow along exactly **if you replace the first two lines of code with these two lines:**

```
import sframe
products = sframe.SFrame('amazon_baby_subset.gl/')
```

Also, replace the line

```
table = graphlab.SFrame({'word': ['(intercept)'] + important_words})
```

with

```
table = sframe.SFrame({'word': ['(intercept)'] + important_words})
```

After these modifications, **you can follow the rest of the IPython notebook and disregard the rest of this reading.**

Note: To install SFrame (without installing GraphLab Create), run

```
pip install sframe
```

If you are NOT using SFrame

Load and process review dataset

1. For this assignment, we will use the same subset of the Amazon product review dataset that we used in Module 3 assignment. The subset was chosen to contain similar numbers of positive and negative reviews, as the original dataset consisted of mostly positive reviews.

Load the dataset into a data frame named **products**.

2. Just like we did previously, we will work with a hand-curated list of important words extracted from the review data. We will also perform 2 simple data transformations:

- Remove punctuation
- Compute word counts (only for the **important_words**)

We start with the first item as follows:

- If your tool supports it, fill n/a values in the review column with empty strings. The n/a values indicate empty reviews. For instance, Pandas's the fillna() method lets you replace all N/A's in the review columns as follows:

```
products = products.fillna({'review':''})  # fill in N/A's in the review column
```

- Write a function **remove_punctuation** that takes a line of text and removes all punctuation from that text. The function should be analogous to the following Python code:

```
def remove_punctuation(text):
    import string
    return text.translate(None, string.punctuation)
```

- Apply the **remove_punctuation** function on every element of the **review** column and assign the result to the new column **review_clean**. **Note.** Many data frame packages support **apply** operation for this type of task. Consult appropriate manuals.

3. Now we proceed with the second item. For each word in **important_words**, we compute a count for the number of times the word occurs in the review. We will store this count in a separate column (one for each word). The result of this feature processing is a single column for each word in **important_words** which keeps a count of the number of times the respective word occurs in the review text.

Note: There are several ways of doing this. One way is to create an anonymous function that counts the occurrence of a particular word and apply it to every element in the **review_clean** column. Repeat this step for every word in **important_words**. Your code should be analogous to the following:

```
for word in important_words:
    products[word] = products['review_clean'].apply(lambda s : s.split().count(word))
```

4. After #2 and #3, the data frame **products** should contain one column for each of the 193 **important_words**. As an example, the column **perfect** contains a count of the number of times the word **prefect** occurs in each of the reviews.

Train-Validation split

5. We split the data into a train-validation split with 80% of the data in the training set and 20% of the data in the validation set. We use seed=2 so that everyone gets the same result. Call the training and validation sets **train_data** and **validation_data**, respectively.

Note: In previous assignments, we have called this a **train-test split**. However, the portion of data that we don't train on will be used to help **select model parameters** (this is known as model selection). Thus, this portion of data should be called a **validation set**. Recall that examining performance of various potential models (i.e. models with different parameters) should be on validation set, while evaluation of the final selected model should always be on test data. Typically, we would also save a portion of the data (a real test set) to test our final model on or use cross-validation on the training set to select our final model. But for the learning purposes of this assignment, we won't do that.

```
train_data, validation_data = products.random_split(.8, seed=2)
```

If you are not using SFrame, download the list of indices for the training and validation sets: module-4-assignment-train-idx.json, module-4-assignment-validation-idx.json. IMPORTANT: If you are using a programming language with 1-based indexing (e.g. R, Matlab), make sure to increment all indices by 1.

Convert data frame to multi-dimensional array

6. Convert **train_data** and **validation_data** into multi-dimensional arrays.

Using the function given in #8 of Module 3 assignment, extract two arrays **feature_matrix_train** and **sentiment_train** from **train_data**. The 2D array **feature_matrix_train** would contain the content of the columns given by the list **important_words**. The 1D array **sentiment_train** would contain the content of the column **sentiment**. Do the same for **validation_data**, producing the arrays **feature_matrix_valid** and **sentiment_valid**. The code should be analogous to this cell:

```
feature_matrix_train, sentiment_train = get_numpy_data(train_data, important_words, 'sentimen
t')
feature_matrix_valid, sentiment_valid = get_numpy_data(validation_data, important_words, 'sen
timent')
```

Building on logistic regression with no L2 penalty assignment

7. Let us now build on the assignment of the previous module. Recall from lecture that the link function for logistic regression can be defined as:

$$P(y_i = +1|\mathbf{x}_i,\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{w}^T h(\mathbf{x}_i))}$$

where the feature vector h(x_i) is given by the word counts of **important_words** in the review x_i.

We will use the **same code** as in this past assignment to make probability predictions since this part is not affected by the L2 penalty. (Only the way in which the coefficients are learned is affected by the addition of a regularization term.) Refer to #10 of Module 3 assignment in order to obtain the function **predict_probability**.

Adding L2 penalty

8. Let us now work on extending logistic regression with an L2 penalty. As discussed in the lectures, the L2 regularization is particularly useful in preventing overfitting. In this assignment, we will explore L2 regularization in detail.

Recall from lecture and the previous assignment that for logistic regression without an L2 penalty, the derivative of the log-likelihood function is:

$$\frac{\partial \ell}{\partial w_j} = \sum_{i=1}^{N} h_j(\mathbf{x}_i)\left(\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i,\mathbf{w})\right)$$

Adding L2 penalty to the derivative

9. It takes only a small modification to add a L2 penalty. All terms indicated in **red** refer to terms that were added due to an **L2 penalty**.

- Recall from the lecture that the link function is still the sigmoid:

$$P(y_i = +1|\mathbf{x}_i,\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{w}^T h(\mathbf{x}_i))}$$

- We add the L2 penalty term to the per-coefficient derivative of log likelihood:

$$\frac{\partial \ell}{\partial w_j} = \sum_{i=1}^{N} h_j(\mathbf{x}_i)\left(\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i,\mathbf{w})\right) - 2\lambda w_j$$

The **per-coefficient derivative for logistic regression with an L2 penalty** is as follows:

$$\frac{\partial \ell}{\partial w_j} = \sum_{i=1}^{N} h_j(\mathbf{x}_i)\left(\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i,\mathbf{w})\right) - 2\lambda w_j$$

and for the intercept term, we have

$$\frac{\partial \ell}{\partial w_0} = \sum_{i=1}^{N} h_0(\mathbf{x}_i)\left(\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i,\mathbf{w})\right)$$

Write a function that computes the derivative of log likelihood with respect to a single coefficient w_j. Unlike its counterpart in the last assignment, the function accepts five parameters:

- **errors**: vector whose i-th value contains

$$\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i,\mathbf{w})$$

- **feature**: vector whose i-th value contains

$$h_j(\mathbf{x}_i)$$

- **coefficient**: the current value of the j-th coefficient.
- **l2_penalty**: the L2 penalty constant λ
- **feature_is_constant**: a Boolean value indicating whether the j-th feature is constant or not.

The function should do the following:

- Take the five parameters as above.
- Compute the dot product of **errors** and **feature** and save the result to **derivative**.
- If **feature_is_constant** is False, subtract the L2 penalty term from **derivative**. Otherwise, do nothing.
- Return **derivative**.

The function should be analogous to the following Python function:

```
def feature_derivative_with_L2(errors, feature, coefficient, l2_penalty, feature_is_constant)
:

    # Compute the dot product of errors and feature
    ## YOUR CODE HERE
    derivative = ...

    # add L2 penalty term for any feature that isn't the intercept.
    if not feature_is_constant:
        ## YOUR CODE HERE
        ...

    return derivative
```

Quiz question: In the code above, was the intercept term regularized?

10. To verify the correctness of the gradient descent algorithm, we write a function for computing log likelihood (which we recall from the last assignment was a topic detailed in an advanced optional video, and used here for its numerical stability), which is given by the formula

$$\ell\ell(\mathbf{w}) = \sum_{i=1}^{N}\Big((\mathbf{1}[y_i = +1] - 1)\mathbf{w}^T h(\mathbf{x}_i) - \ln\big(1+\exp(-\mathbf{w}^T h(\mathbf{x}_i))\big)\Big) - \lambda\|\mathbf{w}\|_2^2$$

The function should be analogous to the following Python function:

```
def compute_log_likelihood_with_L2(feature_matrix, sentiment, coefficients, l2_penalty):
    indicator = (sentiment==+1)
    scores = np.dot(feature_matrix, coefficients)

    lp = np.sum((indicator-1)*scores - np.log(1. + np.exp(-scores))) - l2_penalty*np.sum(coef
ficients[1:]**2)

    return lp
```

Quiz question: Does the term with L2 regularization increase or decrease ℓℓ(w)?

11. The logistic regression function looks almost like the one in the last assignment, with a minor modification to account for the L2 penalty.

Write a function **logistic_regression_with_L2** to fit a logistic regression model under L2 regularization.

The function accepts the following parameters:

- **feature_matrix**: 2D array of features
- **sentiment**: 1D array of class labels
- **initial_coefficients**: 1D array containing initial values of coefficients
- **step_size**: a parameter controlling the size of the gradient steps
- **l2_penalty**: the L2 penalty constant λ
- **max_iter**: number of iterations to run gradient ascent

The function returns the last set of coefficients after performing gradient ascent.

The function carries out the following steps:

1. Initialize vector **coefficients** to **initial_coefficients**.
2. Predict the class probability P(y_i = +1 | x_i,w) using your **predict_probability** function and save it to variable **predictions**.
3. Compute indicator value for (y_i = +1) by comparing **sentiment** against +1. Save it to variable **indicator**.
4. Compute the errors as difference between **indicator** and **predictions**. Save the errors to variable **errors**.
5. For each j-th coefficient, compute the per-coefficient derivative by calling **feature_derivative_L2** with the j-th column of **feature_matrix**. Don't forget to supply the L2 penalty. Then increment the j-th coefficient by (step_size*derivative).
6. Once in a while, insert code to print out the log likelihood.
7. Repeat steps 2-6 for **max_iter** times.

At the end of day, your code should be analogous to the following Python function (with blanks filled in):

```
def logistic_regression_with_L2(feature_matrix, sentiment, initial_coefficients, step_size, l
2_penalty, max_iter):
    coefficients = np.array(initial_coefficients) # make sure it's a numpy array
    for itr in xrange(max_iter):
        # Predict P(y_i = +1|x_i,w) using your predict_probability() function
        ## YOUR CODE HERE
        predictions = ...

        # Compute indicator value for (y_i = +1)
        indicator = (sentiment==+1)

        # Compute the errors as indicator - predictions
        errors = indicator - predictions
        for j in xrange(len(coefficients)): # loop over each coefficient
            is_intercept = (j == 0)
            # Recall that feature_matrix[:,j] is the feature column associated with coefficie
nts[j].
            # Compute the derivative for coefficients[j]. Save it in a variable called deriva
tive
            ## YOUR CODE HERE
            derivative = ...

            # add the step size times the derivative to the current coefficient
            ## YOUR CODE HERE
            ...

        # Checking whether log likelihood is increasing
        if itr <= 15 or (itr <= 100 and itr % 10 == 0) or (itr <= 1000 and itr % 100 == 0) \
        or (itr <= 10000 and itr % 1000 == 0) or itr % 10000 == 0:
            lp = compute_log_likelihood_with_L2(feature_matrix, sentiment, coefficients, l2_p
enalty)
            print 'iteration %*d: log likelihood of observed labels = %.8f' % \
                (int(np.ceil(np.log10(max_iter))), itr, lp)
    return coefficients
```

Explore effects of L2 regularization

12. Now that we have written up all the pieces needed for an L2 solver with logistic regression, let's explore the benefits of using **L2 regularization** while analyzing sentiment for product reviews. **As iterations pass, the log likelihood should increase**.

Let us train models with increasing amounts of regularization, starting with no L2 penalty, which is equivalent to our previous logistic regression implementation. Train 6 models with L2 penalty values 0, 4, 10, 1e2, 1e3, and 1e5. Use the following values for the other parameters:

- **feature_matrix** = **feature_matrix_train** extracted in #7
- **sentiment** = **sentiment_train** extracted in #7
- **initial_coefficients** = a 194-dimensional vector filled with zeros
- **step_size** = 5e-6
- **max_iter** = 501

Save the 6 sets of coefficients as **coefficients_0_penalty**, **coefficients_4_penalty**, **coefficients_10_penalty**, **coefficients_1e2_penalty**, **coefficients_1e3_penalty**, and **coefficients_1e5_penalty** respectively.

Compare coefficients

13. We now compare the **coefficients** for each of the models that were trained above. Create a table of features and learned coefficients associated with each of the different L2 penalty values.

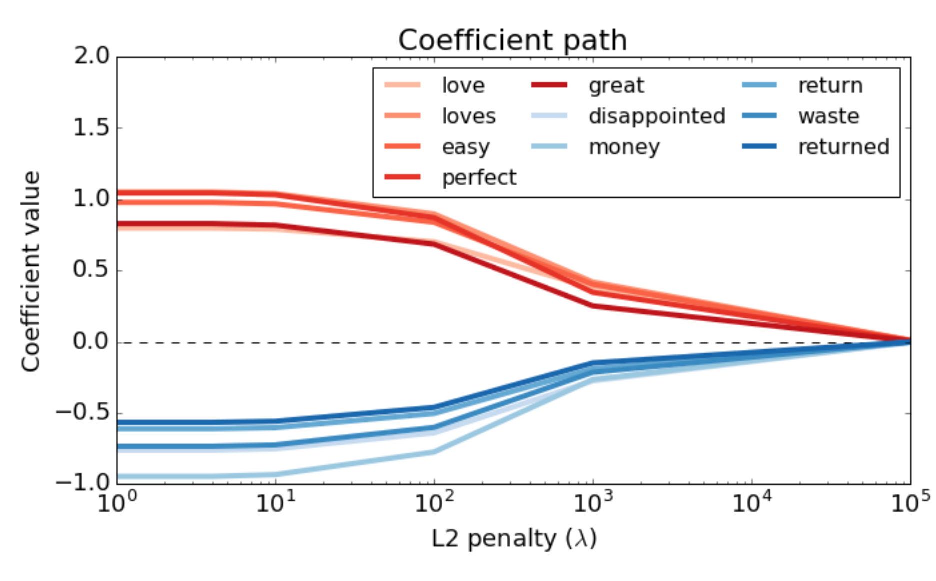
Using **the coefficients trained with L2 penalty 0**, find the 5 most positive words (with largest positive coefficients). Save them to **positive_words**. Similarly, find the 5 most negative words (with largest negative coefficients) and save them to **negative_words**.

Quiz Question. Which of the following is **not** listed in either **positive_words** or **negative_words**?

14. Let us observe the effect of increasing L2 penalty on the 10 words just selected. Make a plot of the coefficients for the 10 words over the different values of L2 penalty.

Hints:

- First, extract rows corresponding to **positive_words**. Do the same for **negative_words**.
- Then plot each of the extracted rows. The x axis should be L2 penalty and the y axis should be the coefficient value.
- Use log scale for the x axis, as the L2 penalty values are exponentially spaced.



If you are using Python, you can use matplotlib to generate the plot.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = 10, 6

def make_coefficient_plot(table, positive_words, negative_words, l2_penalty_list):
    cmap_positive = plt.get_cmap('Reds')
    cmap_negative = plt.get_cmap('Blues')

    xx = l2_penalty_list
    plt.plot(xx, [0.]*len(xx), '--', lw=1, color='k')

    table_positive_words = table[table['word'].isin(positive_words)]
    table_negative_words = table[table['word'].isin(negative_words)]
    del table_positive_words['word']
    del table_negative_words['word']

    for i in xrange(len(positive_words)):
        color = cmap_positive(0.8*((i+1)/(len(positive_words)*1.2)+0.15))
        plt.plot(xx, table_positive_words[i:i+1].as_matrix().flatten(),
                 '-', label=positive_words[i], linewidth=4.0, color=color)

    for i in xrange(len(negative_words)):
        color = cmap_negative(0.8*((i+1)/(len(negative_words)*1.2)+0.15))
        plt.plot(xx, table_negative_words[i:i+1].as_matrix().flatten(),
                 '-', label=negative_words[i], linewidth=4.0, color=color)

    plt.legend(loc='best', ncol=3, prop={'size':16}, columnspacing=0.5)
    plt.axis([1, 1e5, -1, 2])
    plt.title('Coefficient path')
    plt.xlabel('L2 penalty ($\lambda$)')
    plt.ylabel('Coefficient value')
    plt.xscale('log')
    plt.rcParams.update({'font.size': 18})
    plt.tight_layout()


make_coefficient_plot(table, positive_words, negative_words, l2_penalty_list=[0, 4, 10, 1e2,
1e3, 1e5])
```

Quiz Question: (True/False) All coefficients consistently get smaller in size as L2 penalty is increased.

Quiz Question: (True/False) Relative order of coefficients is preserved as L2 penalty is increased. (If word 'cat' was more positive than word 'dog', then it remains to be so as L2 penalty is increased.)

Measuring accuracy

15. Now, let us compute the accuracy of the classifier model. Recall that the accuracy is given by

$$\text{accuracy} = \frac{\#\text{ correctly classified data points}}{\#\text{ total data points}}$$

Recall from lecture that that the class prediction is calculated using

$$\hat{y}_i = \begin{cases} +1 & h(\mathbf{x}_i)^T\mathbf{w} > 0 \\ -1 & h(\mathbf{x}_i)^T\mathbf{w} \le 0 \end{cases}$$

Note: It is important to know that the model prediction code doesn't change even with L2 penalty. The only thing that changes is the estimated coefficients used in this prediction are different with L2 penalty.

- **Quiz question**: Which model (L2 = 0, 4, 10, 100, 1e3, 1e5) has the **highest** accuracy on the **training** data?
- **Quiz question**: Which model (L2 = 0, 4, 10, 100, 1e3, 1e5) has the **highest** accuracy on the **validation** data?
- **Quiz question**: Does the **highest** accuracy on the **training** data imply that the model is the best one?