Logistic Regression with L2 regularization

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

- Extract features from Amazon product reviews.
- Convert an SFrame 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.

Fire up GraphLab Create

Make sure you have the latest version of GraphLab Create. Upgrade by

```
pip install graphlab-create --upgrade
```

See this page (https://dato.com/download/) for detailed instructions on upgrading.

In [1]:

```
from __future__ import division
import graphlab
```

Load and process review dataset

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.

In [2]:

```
products = graphlab.SFrame('amazon_baby_subset.gl/')
```

```
This non-commercial license of GraphLab Create for academic use is assigned
to amitha353@gmail.com and will expire on May 07, 2019.

[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging:
C:\Users\Amitha\AppData\Local\Temp\graphlab_server_1532692366.log.0
```

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:

1. Remove punctuation using Python's built-in (https://docs.python.org/2/library/string.html) string functionality.
2. Compute word counts (only for the **important_words**)

Refer to Module 3 assignment for more details.

In [3]:

```
# The same feature processing (same as the previous assignments)
# ---------------------------------------------------------------
import json
with open('important_words.json', 'r') as f: # Reads the list of most frequent words
    important_words = json.load(f)
important_words = [str(s) for s in important_words]


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

# Remove punctuation.
products['review_clean'] = products['review'].apply(remove_punctuation)

# Split out the words into individual columns
for word in important_words:
    products[word] = products['review_clean'].apply(lambda s : s.split().count(word))
```

Now, let us take a look at what the dataset looks like (**Note:** This may take a few minutes).

In [4]:

```
products
```

Out[4]:

name	review	rating	sentiment	review_clean	baby
Stop Pacifier Sucking without tears with ...	All of my kids have cried non-stop when I tried to ...	5.0	1	All of my kids have cried nonstop when I tried to ...	0
Nature's Lullabies Second Year Sticker Calendar ...	We wanted to get something to keep track ...	5.0	1	We wanted to get something to keep track ...	0
Nature's Lullabies Second Year Sticker Calendar ...	My daughter had her 1st baby over a year ago. ...	5.0	1	My daughter had her 1st baby over a year ago She ...	1
Lamaze Peekaboo, I Love You ...	One of baby's first and favorite books, and i ...	4.0	1	One of babys first and favorite books and it is ...	0
SoftPlay Peek-A-Boo Where's Elmo A Childr ...	Very cute interactive book! My son loves this ...	5.0	1	Very cute interactive book My son loves this ...	0
Our Baby Girl Memory Book	Beautiful book, I love it to record cherished t ...	5.0	1	Beautiful book I love it to record cherished t ...	0
Hunnt&reg; Falling Flowers and Birds Kids ...	Try this out for a spring project !Easy ,fun and ...	5.0	1	Try this out for a spring project Easy fun and ...	0
Blessed By Pope Benedict XVI Divine Mercy Full ...	very nice Divine Mercy Pendant of Jesus now on ...	5.0	1	very nice Divine Mercy Pendant of Jesus now on ...	0

Cloth Diaper	We bought	4.0	1	We bought	0

Train-Validation split

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.

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**. 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 a validation set, while evaluation of selected model should always be on a test set.

In [5]:

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

print 'Training set   : %d data points' % len(train_data)
print 'Validation set : %d data points' % len(validation_data)
```

```
Training set   : 42361 data points
Validation set : 10711 data points
```

Convert SFrame to NumPy array

Just like in the second assignment of the previous module, we provide you with a function that extracts columns from an SFrame and converts them into a NumPy array. Two arrays are returned: one representing features and another representing class labels.

Note: The feature matrix includes an additional column 'intercept' filled with 1's to take account of the intercept term.

In [6]:

```
import numpy as np

def get_numpy_data(data_sframe, features, label):
    data_sframe['intercept'] = 1
    features = ['intercept'] + features
    features_sframe = data_sframe[features]
    feature_matrix = features_sframe.to_numpy()
    label_sarray = data_sframe[label]
    label_array = label_sarray.to_numpy()
    return(feature_matrix, label_array)
```

We convert both the training and validation sets into NumPy arrays.

Warning: This may take a few minutes.

In [7]:

```
feature_matrix_train, sentiment_train = get_numpy_data(train_data, important_words, 'sentim
feature_matrix_valid, sentiment_valid = get_numpy_data(validation_data, important_words, 's
```

Are you running this notebook on an Amazon EC2 t2.micro instance? (If you are using your own machine, please skip this section)

It has been reported that t2.micro instances do not provide sufficient power to complete the conversion in acceptable amount of time. For interest of time, please refrain from running `get_numpy_data` function. Instead, download the binary file (https://s3.amazonaws.com/static.dato.com/files/coursera/course-3/numpy-arrays/module-4-assignment-numpy-arrays.npz) containing the four NumPy arrays you'll need for the assignment. To load the arrays, run the following commands:

```
    arrays = np.load('module-4-assignment-numpy-arrays.npz')
    feature_matrix_train, sentiment_train = arrays['feature_matrix_train'], arrays['se
    ntiment_train']
    feature_matrix_valid, sentiment_valid = arrays['feature_matrix_valid'], arrays['se
    ntiment_valid']
```

Building on logistic regression with no L2 penalty assignment

Let us now build on Module 3 assignment. 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 $h(\mathbf{x}_i)$ is given by the word counts of **important_words** in the review $\mathbf{x}_i$.

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.)

In [8]:

```
'''
produces probablistic estimate for P(y_i = +1 | x_i, w).
estimate ranges between 0 and 1.
'''
def predict_probability(feature_matrix, coefficients):
    # Take dot product of feature_matrix and coefficients
    ## YOUR CODE HERE
    scores = np.dot(feature_matrix, coefficients)

    # Compute P(y_i = +1 | x_i, w) using the link function
    ## YOUR CODE HERE
    predictions = 1/(1 + np.exp(-scores))

    return predictions
```

Adding L2 penalty

Let us now work on extending logistic regression with L2 regularization. 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

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

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

Note: As we did in the Regression course, we do not apply the L2 penalty on the intercept. A large intercept does not necessarily indicate overfitting because the intercept is not associated with any particular feature.

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 arguments:

- `errors` vector containing $(\mathbf{1}[y_i = +1] - P(y_i = +1|\mathbf{x}_i, \mathbf{w}))$ for all i
- `feature` vector containing $h_j(\mathbf{x}_i)$ for all i
- `coefficient` containing the current value of coefficient w_j.
- `l2_penalty` representing the L2 penalty constant λ
- `feature_is_constant` telling whether the j-th feature is constant or not.

In [9]:

```
def feature_derivative_with_L2(errors, feature, coefficient, l2_penalty, feature_is_constar

    # Compute the dot product of errors and feature
    ## YOUR CODE HERE
    derivative = np.dot(errors, feature)

    # add L2 penalty term for any feature that isn't the intercept.
    if not feature_is_constant:
        ## YOUR CODE HERE
        derivative -= (2 * l2_penalty * coefficient)

    return derivative
```

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

To verify the correctness of the gradient ascent algorithm, we provide 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).

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

In [10]:

```
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(cc

    return lp
```

Quiz Question: Does the term with L2 regularization increase or decrease $\ell\ell(\mathbf{w})$?

The logistic regression function looks almost like the one in the last assignment, with a minor modification to account for the L2 penalty. Fill in the code below to complete this modification.

In [11]:

```
def logistic_regression_with_L2(feature_matrix, sentiment, initial_coefficients, step_size,
    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 = predict_probability(feature_matrix, coefficients)

        # 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 coeffic
            # Compute the derivative for coefficients[j]. Save it in a variable called deri
            ## YOUR CODE HERE
            derivative = feature_derivative_with_L2(errors, feature_matrix[:,j], coefficier

            # add the step size times the derivative to the current coefficient
            ## YOUR CODE HERE
            coefficients[j] += step_size * derivative

        # 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
            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

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

Below, we train models with increasing amounts of regularization, starting with no L2 penalty, which is equivalent to our previous logistic regression implementation.

In [12]:

```
# run with L2 = 0
coefficients_0_penalty = logistic_regression_with_L2(feature_matrix_train, sentiment_train,
                                                     initial_coefficients=np.zeros(194),
                                                     step_size=5e-6, l2_penalty=0, max_iter
```

```
iteration   0: log likelihood of observed labels = -29179.39138303
iteration   1: log likelihood of observed labels = -29003.71259047
iteration   2: log likelihood of observed labels = -28834.66187288
iteration   3: log likelihood of observed labels = -28671.70781507
iteration   4: log likelihood of observed labels = -28514.43078198
iteration   5: log likelihood of observed labels = -28362.48344665
iteration   6: log likelihood of observed labels = -28215.56713122
iteration   7: log likelihood of observed labels = -28073.41743783
iteration   8: log likelihood of observed labels = -27935.79536396
iteration   9: log likelihood of observed labels = -27802.48168669
iteration  10: log likelihood of observed labels = -27673.27331484
iteration  11: log likelihood of observed labels = -27547.98083656
iteration  12: log likelihood of observed labels = -27426.42679977
iteration  13: log likelihood of observed labels = -27308.44444728
iteration  14: log likelihood of observed labels = -27193.87673876
iteration  15: log likelihood of observed labels = -27082.57555831
iteration  20: log likelihood of observed labels = -26570.43059938
iteration  30: log likelihood of observed labels = -25725.48742389
iteration  40: log likelihood of observed labels = -25055.53326910
iteration  50: log likelihood of observed labels = -24509.63590026
iteration  60: log likelihood of observed labels = -24054.97906083
iteration  70: log likelihood of observed labels = -23669.51640848
iteration  80: log likelihood of observed labels = -23337.89167628
iteration  90: log likelihood of observed labels = -23049.07066021
iteration 100: log likelihood of observed labels = -22794.90974921
iteration 200: log likelihood of observed labels = -21283.29527353
iteration 300: log likelihood of observed labels = -20570.97485473
iteration 400: log likelihood of observed labels = -20152.21466944
iteration 500: log likelihood of observed labels = -19876.62333410
```

In [13]:

```
# run with L2 = 4
coefficients_4_penalty = logistic_regression_with_L2(feature_matrix_train, sentiment_train,
                                                      initial_coefficients=np.zeros(194),
                                                      step_size=5e-6, l2_penalty=4, max_ite
```

```
iteration   0: log likelihood of observed labels = -29179.39508175
iteration   1: log likelihood of observed labels = -29003.73417180
iteration   2: log likelihood of observed labels = -28834.71441858
iteration   3: log likelihood of observed labels = -28671.80345068
iteration   4: log likelihood of observed labels = -28514.58077957
iteration   5: log likelihood of observed labels = -28362.69830317
iteration   6: log likelihood of observed labels = -28215.85663259
iteration   7: log likelihood of observed labels = -28073.79071393
iteration   8: log likelihood of observed labels = -27936.26093762
iteration   9: log likelihood of observed labels = -27803.04751805
iteration  10: log likelihood of observed labels = -27673.94684207
iteration  11: log likelihood of observed labels = -27548.76901327
iteration  12: log likelihood of observed labels = -27427.33612958
iteration  13: log likelihood of observed labels = -27309.48101569
iteration  14: log likelihood of observed labels = -27195.04624253
iteration  15: log likelihood of observed labels = -27083.88333261
iteration  20: log likelihood of observed labels = -26572.49874392
iteration  30: log likelihood of observed labels = -25729.32604153
iteration  40: log likelihood of observed labels = -25061.34245801
iteration  50: log likelihood of observed labels = -24517.52091982
iteration  60: log likelihood of observed labels = -24064.99093939
iteration  70: log likelihood of observed labels = -23681.67373669
iteration  80: log likelihood of observed labels = -23352.19298741
iteration  90: log likelihood of observed labels = -23065.50180166
iteration 100: log likelihood of observed labels = -22813.44844580
iteration 200: log likelihood of observed labels = -21321.14164794
iteration 300: log likelihood of observed labels = -20624.98634439
iteration 400: log likelihood of observed labels = -20219.92048845
iteration 500: log likelihood of observed labels = -19956.11341777
```

In [14]:

```
# run with L2 = 10
coefficients_10_penalty = logistic_regression_with_L2(feature_matrix_train, sentiment_trair
                                                       initial_coefficients=np.zeros(194),
                                                       step_size=5e-6, l2_penalty=10, max_it
```

```
iteration   0: log likelihood of observed labels = -29179.40062984
iteration   1: log likelihood of observed labels = -29003.76654163
iteration   2: log likelihood of observed labels = -28834.79322654
iteration   3: log likelihood of observed labels = -28671.94687528
iteration   4: log likelihood of observed labels = -28514.80571589
iteration   5: log likelihood of observed labels = -28363.02048079
iteration   6: log likelihood of observed labels = -28216.29071186
iteration   7: log likelihood of observed labels = -28074.35036891
iteration   8: log likelihood of observed labels = -27936.95892966
iteration   9: log likelihood of observed labels = -27803.89576265
iteration  10: log likelihood of observed labels = -27674.95647005
iteration  11: log likelihood of observed labels = -27549.95042714
iteration  12: log likelihood of observed labels = -27428.69905549
iteration  13: log likelihood of observed labels = -27311.03455140
iteration  14: log likelihood of observed labels = -27196.79890162
iteration  15: log likelihood of observed labels = -27085.84308528
iteration  20: log likelihood of observed labels = -26575.59697506
iteration  30: log likelihood of observed labels = -25735.07304608
iteration  40: log likelihood of observed labels = -25070.03447306
iteration  50: log likelihood of observed labels = -24529.31188025
iteration  60: log likelihood of observed labels = -24079.95349572
iteration  70: log likelihood of observed labels = -23699.83199186
iteration  80: log likelihood of observed labels = -23373.54108747
iteration  90: log likelihood of observed labels = -23090.01500055
iteration 100: log likelihood of observed labels = -22841.08995135
iteration 200: log likelihood of observed labels = -21377.25595328
iteration 300: log likelihood of observed labels = -20704.63995428
iteration 400: log likelihood of observed labels = -20319.25685307
iteration 500: log likelihood of observed labels = -20072.16321721
```

In [15]:

```
# run with L2 = 1e2
coefficients_1e2_penalty = logistic_regression_with_L2(feature_matrix_train, sentiment_trai
                                                       initial_coefficients=np.zeros(194),
                                                       step_size=5e-6, l2_penalty=1e2, max_
```

```
iteration   0: log likelihood of observed labels = -29179.48385120
iteration   1: log likelihood of observed labels = -29004.25177457
iteration   2: log likelihood of observed labels = -28835.97382190
iteration   3: log likelihood of observed labels = -28674.09410083
iteration   4: log likelihood of observed labels = -28518.17112932
iteration   5: log likelihood of observed labels = -28367.83774654
iteration   6: log likelihood of observed labels = -28222.77708939
iteration   7: log likelihood of observed labels = -28082.70799392
iteration   8: log likelihood of observed labels = -27947.37595368
iteration   9: log likelihood of observed labels = -27816.54738615
iteration  10: log likelihood of observed labels = -27690.00588850
iteration  11: log likelihood of observed labels = -27567.54970126
iteration  12: log likelihood of observed labels = -27448.98991327
iteration  13: log likelihood of observed labels = -27334.14912742
iteration  14: log likelihood of observed labels = -27222.86041863
iteration  15: log likelihood of observed labels = -27114.96648229
iteration  20: log likelihood of observed labels = -26621.50201299
iteration  30: log likelihood of observed labels = -25819.72803950
iteration  40: log likelihood of observed labels = -25197.34035501
iteration  50: log likelihood of observed labels = -24701.03698195
iteration  60: log likelihood of observed labels = -24296.66378580
iteration  70: log likelihood of observed labels = -23961.38842316
iteration  80: log likelihood of observed labels = -23679.38088853
iteration  90: log likelihood of observed labels = -23439.31824267
iteration 100: log likelihood of observed labels = -23232.88192018
iteration 200: log likelihood of observed labels = -22133.50726528
iteration 300: log likelihood of observed labels = -21730.03957488
iteration 400: log likelihood of observed labels = -21545.87572145
iteration 500: log likelihood of observed labels = -21451.95551390
```

In [16]:

```
# run with L2 = 1e3
coefficients_1e3_penalty = logistic_regression_with_L2(feature_matrix_train, sentiment_trai
                                                       initial_coefficients=np.zeros(194),
                                                       step_size=5e-6, l2_penalty=1e3, max_
```

```
iteration   0: log likelihood of observed labels = -29180.31606471
iteration   1: log likelihood of observed labels = -29009.07176112
iteration   2: log likelihood of observed labels = -28847.62378912
iteration   3: log likelihood of observed labels = -28695.14439397
iteration   4: log likelihood of observed labels = -28550.95060743
iteration   5: log likelihood of observed labels = -28414.45771129
iteration   6: log likelihood of observed labels = -28285.15124375
iteration   7: log likelihood of observed labels = -28162.56976044
iteration   8: log likelihood of observed labels = -28046.29387744
iteration   9: log likelihood of observed labels = -27935.93902900
iteration  10: log likelihood of observed labels = -27831.15045502
iteration  11: log likelihood of observed labels = -27731.59955260
iteration  12: log likelihood of observed labels = -27636.98108219
iteration  13: log likelihood of observed labels = -27547.01092670
iteration  14: log likelihood of observed labels = -27461.42422295
iteration  15: log likelihood of observed labels = -27379.97375625
iteration  20: log likelihood of observed labels = -27027.18208317
iteration  30: log likelihood of observed labels = -26527.22737267
iteration  40: log likelihood of observed labels = -26206.59048765
iteration  50: log likelihood of observed labels = -25995.96903148
iteration  60: log likelihood of observed labels = -25854.95710284
iteration  70: log likelihood of observed labels = -25759.08109950
iteration  80: log likelihood of observed labels = -25693.05688014
iteration  90: log likelihood of observed labels = -25647.09929349
iteration 100: log likelihood of observed labels = -25614.81468705
iteration 200: log likelihood of observed labels = -25536.20998919
iteration 300: log likelihood of observed labels = -25532.57691220
iteration 400: log likelihood of observed labels = -25532.35543765
iteration 500: log likelihood of observed labels = -25532.33970049
```

In [17]:

```
# run with L2 = 1e5
coefficients_1e5_penalty = logistic_regression_with_L2(feature_matrix_train, sentiment_trai
                                                       initial_coefficients=np.zeros(194),
                                                       step_size=5e-6, l2_penalty=1e5, max_
```

```
iteration   0: log likelihood of observed labels = -29271.85955115
iteration   1: log likelihood of observed labels = -29271.71006589
iteration   2: log likelihood of observed labels = -29271.65738833
iteration   3: log likelihood of observed labels = -29271.61189923
iteration   4: log likelihood of observed labels = -29271.57079975
iteration   5: log likelihood of observed labels = -29271.53358505
iteration   6: log likelihood of observed labels = -29271.49988440
iteration   7: log likelihood of observed labels = -29271.46936584
iteration   8: log likelihood of observed labels = -29271.44172890
iteration   9: log likelihood of observed labels = -29271.41670149
iteration  10: log likelihood of observed labels = -29271.39403722
iteration  11: log likelihood of observed labels = -29271.37351294
iteration  12: log likelihood of observed labels = -29271.35492661
iteration  13: log likelihood of observed labels = -29271.33809523
iteration  14: log likelihood of observed labels = -29271.32285309
iteration  15: log likelihood of observed labels = -29271.30905015
iteration  20: log likelihood of observed labels = -29271.25729150
iteration  30: log likelihood of observed labels = -29271.20657205
iteration  40: log likelihood of observed labels = -29271.18775997
iteration  50: log likelihood of observed labels = -29271.18078247
iteration  60: log likelihood of observed labels = -29271.17819447
iteration  70: log likelihood of observed labels = -29271.17723457
iteration  80: log likelihood of observed labels = -29271.17687853
iteration  90: log likelihood of observed labels = -29271.17674648
iteration 100: log likelihood of observed labels = -29271.17669750
iteration 200: log likelihood of observed labels = -29271.17666862
iteration 300: log likelihood of observed labels = -29271.17666862
iteration 400: log likelihood of observed labels = -29271.17666862
iteration 500: log likelihood of observed labels = -29271.17666862
```

Compare coefficients

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

Below is a simple helper function that will help us create this table.

In [18]:

```
table = graphlab.SFrame({'word': ['(intercept)'] + important_words})
def add_coefficients_to_table(coefficients, column_name):
    table[column_name] = coefficients
    return table
```

Now, let's run the function `add_coefficients_to_table` for each of the L2 penalty strengths.

In [19]:

```
add_coefficients_to_table(coefficients_0_penalty, 'coefficients [L2=0]')
add_coefficients_to_table(coefficients_4_penalty, 'coefficients [L2=4]')
add_coefficients_to_table(coefficients_10_penalty, 'coefficients [L2=10]')
add_coefficients_to_table(coefficients_1e2_penalty, 'coefficients [L2=1e2]')
add_coefficients_to_table(coefficients_1e3_penalty, 'coefficients [L2=1e3]')
add_coefficients_to_table(coefficients_1e5_penalty, 'coefficients [L2=1e5]')
```

Out[19]:

word	coefficients [L2=0]	coefficients [L2=4]	coefficients [L2=10]		coefficients [L2=1e3]	coefficients [L2=1e5]
(intercept)	-0.0637421352275	-0.0631430877074	-0.06225594377	-	5.38675326957e-05	0.0113617511844
baby	0.0740730059216	0.0739938541405	0.0738773534804		0.0597516888364	0.0017841492163
one	0.0127525057784	0.0124949704481	0.0121152529534		-0.00876091762004	-0.00182685568023
great	0.801624989778	0.796896933003	0.789935147221		0.376011714222	0.00894956049736
love	1.05855398207	1.05085568099	1.03952851585		0.418353644134	0.0090417372977
use	-0.000104152191248	0.000162857656177	0.000555710975756	(	0.0173264874461	0.000417863934616
would	-0.287021443534	-0.286027202975	-0.284564035562		-0.188662422049	-0.008127027099
like	-0.00338447399293	-0.00344208577045	-0.00352729444966	-	-0.00704307584353	-0.000826650270031
easy	0.984558819873	0.977600149782	0.967361836631		0.401903971363	0.0088076812121
little	0.524419456364	0.521384726107	0.516917392491		0.251220653959	0.00594051364038

[194 rows x 7 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

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**?

In [20]:

```
table[['word','coefficients [L2=0]']].sort('coefficients [L2=0]', ascending = False)[0:5]
```

Out[20]:

word	coefficients [L2=0]
love	1.05855398207
loves	1.0524840478
easy	0.984558819873
perfect	0.835693207638
great	0.801624989778

[5 rows x 2 columns]

In [25]:

```
positive_words = table.sort('coefficients [L2=0]', ascending = False)[0:5]['word']
print positive_words
```

```
['love', 'loves', 'easy', 'perfect', 'great']
```

In [26]:

```
negative_words = table.sort('coefficients [L2=0]', ascending = True)[0:5]['word']
print negative_words
```

```
['disappointed', 'money', 'return', 'waste', 'returned']
```

Let us observe the effect of increasing L2 penalty on the 10 words just selected. We provide you with a utility function to plot the coefficient path.

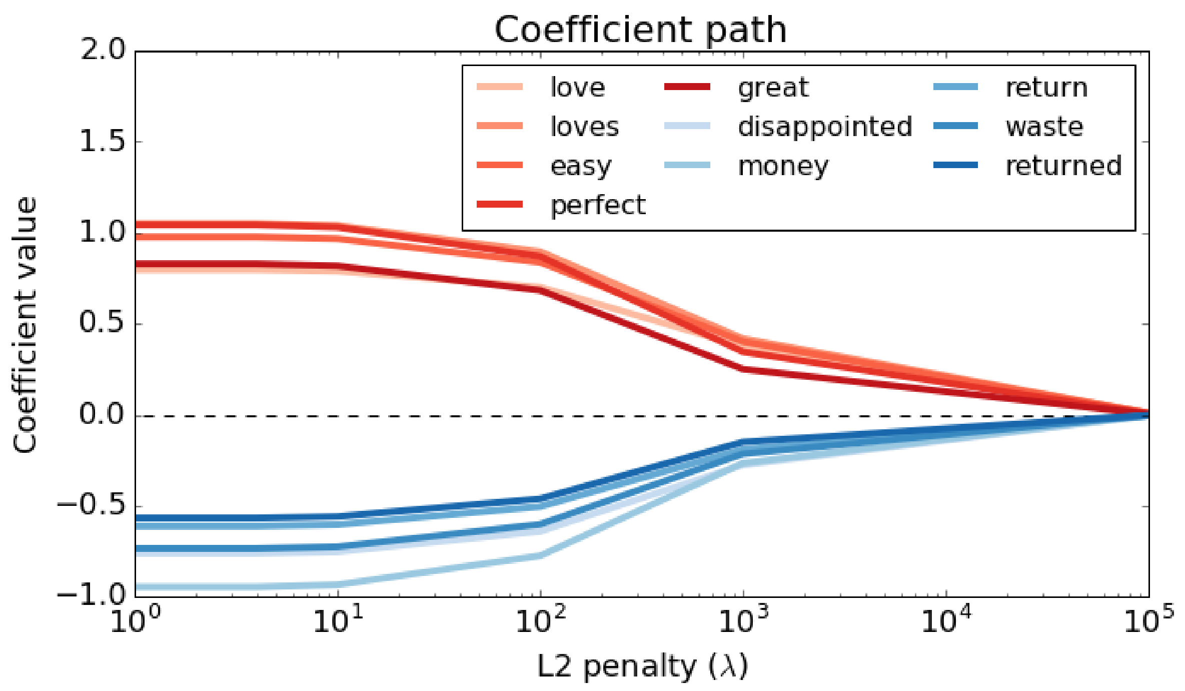
In [27]:

```
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.filter_by(column_name='word', values=positive_words)
    table_negative_words = table.filter_by(column_name='word', values=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].to_numpy().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].to_numpy().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()
```

Run the following cell to generate the plot. Use the plot to answer the following quiz question.

In [28]:

```
make_coefficient_plot(table, positive_words, negative_words, l2_penalty_list=[0, 4, 10, 1e2
```



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

Quiz Question: (True/False) The relative order of coefficients is preserved as the L2 penalty is increased. (For example, if the coefficient for 'cat' was more positive than that for 'dog', this remains true as the L2 penalty increases.)

Measuring accuracy

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

$$\hat{y}_i = \left\{ \begin{array}{ll} +1 & h(\mathbf{x}_i)^T \mathbf{w} > 0 \\ -1 & h(\mathbf{x}_i)^T \mathbf{w} \leq 0 \end{array} \right|$$

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

Based on the above, we will use the same code that was used in Module 3 assignment.

In [29]:

```
def get_classification_accuracy(feature_matrix, sentiment, coefficients):
    scores = np.dot(feature_matrix, coefficients)
    apply_threshold = np.vectorize(lambda x: 1. if x > 0  else -1.)
    predictions = apply_threshold(scores)

    num_correct = (predictions == sentiment).sum()
    accuracy = num_correct / len(feature_matrix)
    return accuracy
```

Below, we compare the accuracy on the **training data** and **validation data** for all the models that were trained in this assignment. We first calculate the accuracy values and then build a simple report summarizing the performance for the various models.

In [30]:

```
train_accuracy = {}
train_accuracy[0]   = get_classification_accuracy(feature_matrix_train, sentiment_train, co
train_accuracy[4]   = get_classification_accuracy(feature_matrix_train, sentiment_train, co
train_accuracy[10]  = get_classification_accuracy(feature_matrix_train, sentiment_train, co
train_accuracy[1e2] = get_classification_accuracy(feature_matrix_train, sentiment_train, co
train_accuracy[1e3] = get_classification_accuracy(feature_matrix_train, sentiment_train, co
train_accuracy[1e5] = get_classification_accuracy(feature_matrix_train, sentiment_train, co

validation_accuracy = {}
validation_accuracy[0]   = get_classification_accuracy(feature_matrix_valid, sentiment_vali
validation_accuracy[4]   = get_classification_accuracy(feature_matrix_valid, sentiment_vali
validation_accuracy[10]  = get_classification_accuracy(feature_matrix_valid, sentiment_vali
validation_accuracy[1e2] = get_classification_accuracy(feature_matrix_valid, sentiment_vali
validation_accuracy[1e3] = get_classification_accuracy(feature_matrix_valid, sentiment_vali
validation_accuracy[1e5] = get_classification_accuracy(feature_matrix_valid, sentiment_vali
```

In [31]:

```
# Build a simple report
for key in sorted(validation_accuracy.keys()):
    print "L2 penalty = %g" % key
    print "train accuracy = %s, validation_accuracy = %s" % (train_accuracy[key], validatic
    print "------------------------------------------------------------------------------
```

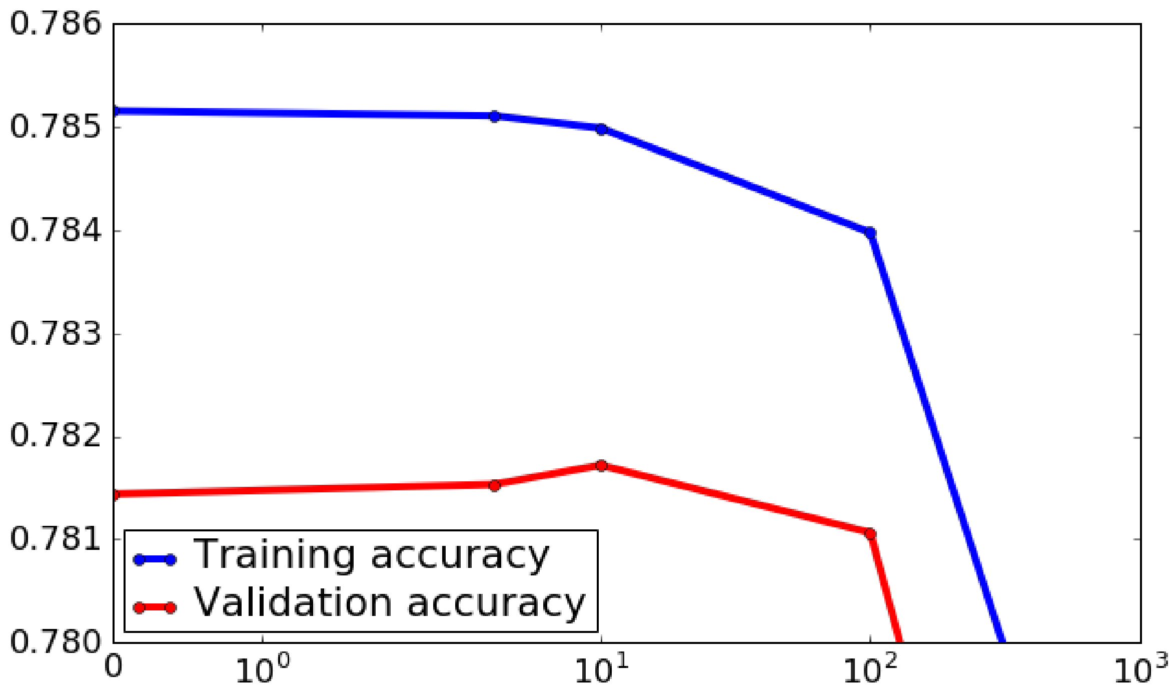
```
L2 penalty = 0
train accuracy = 0.785156157787, validation_accuracy = 0.78143964149
----------------------------------------------------------------------------
----
L2 penalty = 4
train accuracy = 0.785108944548, validation_accuracy = 0.781533003454
----------------------------------------------------------------------------
----
L2 penalty = 10
train accuracy = 0.784990911452, validation_accuracy = 0.781719727383
----------------------------------------------------------------------------
----
L2 penalty = 100
train accuracy = 0.783975826822, validation_accuracy = 0.781066193633
----------------------------------------------------------------------------
----
L2 penalty = 1000
train accuracy = 0.775855149784, validation_accuracy = 0.771356549342
----------------------------------------------------------------------------
----
L2 penalty = 100000
train accuracy = 0.680366374731, validation_accuracy = 0.667818130893
----------------------------------------------------------------------------
----
```

In [32]:

```
# Optional. Plot accuracy on training and validation sets over choice of L2 penalty.
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = 10, 6

sorted_list = sorted(train_accuracy.items(), key=lambda x:x[0])
plt.plot([p[0] for p in sorted_list], [p[1] for p in sorted_list], 'bo-', linewidth=4, labe
sorted_list = sorted(validation_accuracy.items(), key=lambda x:x[0])
plt.plot([p[0] for p in sorted_list], [p[1] for p in sorted_list], 'ro-', linewidth=4, labe
plt.xscale('symlog')
plt.axis([0, 1e3, 0.78, 0.786])
plt.legend(loc='lower left')
plt.rcParams.update({'font.size': 18})
plt.tight_layout
```

Out[32]:

```
<function matplotlib.pyplot.tight_layout>
```



- **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?

Quiz

1. In the function **feature_derivative_with_L2**, was the intercept term regularized?

- Yes
- [✓ No]

2. Does the term with L2 regularization increase or decrease the log likelihood $\ell\ell(\mathbf{w})$?

- Increases
- [✓ Decreases]

3. Which of the following words is **not** listed in either **positive_words** or **negative_words**?

- love
- disappointed
- great
- money
- [✓ quality]

4. Questions 5 and 6 use the coefficient plot of the words in **positive_words** and **negative_words**.

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

- [✓ True]
- False

5. Questions 5 and 6 use the coefficient plot of the words in **positive_words** and **negative_words**.

(True/False) The relative order of coefficients is preserved as the L2 penalty is increased. (For example, if the coefficient for 'cat' was more positive than that for 'dog', this remains true as the L2 penalty increases.)

- True
- [✓ False]

6. Questions 7, 8, and 9 ask you about the 6 models trained with different L2 penalties.

Which of the following models has the **highest** accuracy on the **training** data?

- [✓ Model trained with L2 penalty = 0]
- Model trained with L2 penalty = 4
- Model trained with L2 penalty = 10
- Model trained with L2 penalty = 100
- Model trained with L2 penalty = 1e3
- Model trained with L2 penalty = 1e5

7. Questions 7, 8, and 9 ask you about the 6 models trained with different L2 penalties.

Which of the following models has the **highest** accuracy on the **validation** data?

- Model trained with L2 penalty = 0
- Model trained with L2 penalty = 4
- [✓ Model trained with L2 penalty = 10]
- Model trained with L2 penalty = 100
- Model trained with L2 penalty = 1e3
- Model trained with L2 penalty = 1e5

8. Questions 7, 8, and 9 ask you about the 6 models trained with different L2 penalties.

Does the **highest** accuracy on the **training** data imply that the model is the best one?

- Yes
- [✓ No]