Predicting sentiment from product reviews

The goal of this first notebook is to explore logistic regression and feature engineering with existing GraphLab functions.

In this notebook you will use product review data from Amazon.com to predict whether the sentiments about a product (from its reviews) are positive or negative.

- Use SFrames to do some feature engineering
- Train a logistic regression model to predict the sentiment of product reviews.
- Inspect the weights (coefficients) of a trained logistic regression model.
- Make a prediction (both class and probability) of sentiment for a new product review.
- Given the logistic regression weights, predictors and ground truth labels, write a function to compute the **accuracy** of the model.
- Inspect the coefficients of the logistic regression model and interpret their meanings.
- Compare multiple logistic regression models.

Let's get started!

Fire up GraphLab Create

Make sure you have the latest version of GraphLab Create.

In [4]:

```
from __future__ import division
import graphlab
import math
import string
```

Data preparation

We will use a dataset consisting of baby product reviews on Amazon.com.

In [5]:

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

Now, let us see a preview of what the dataset looks like.

In [6]:

```
products
```

Out[6]:

name	review	rating
Planetwise Flannel Wipes	These flannel wipes are OK, but in my opinion ...	3.0
Planetwise Wipe Pouch	it came early and was not disappointed. i love ...	5.0
Annas Dream Full Quilt with 2 Shams ...	Very soft and comfortable and warmer than it ...	5.0
Stop Pacifier Sucking without tears with ...	This is a product well worth the purchase. I ...	5.0
Stop Pacifier Sucking without tears with ...	All of my kids have cried non-stop when I tried to ...	5.0
Stop Pacifier Sucking without tears with ...	When the Binky Fairy came to our house, we didn't ...	5.0
A Tale of Baby's Days with Peter Rabbit ...	Lovely book, it's bound tightly so you may no ...	4.0
Baby Tracker® - Daily Childcare Journal, ...	Perfect for new parents. We were able to keep ...	5.0
Baby Tracker® - Daily Childcare Journal, ...	A friend of mine pinned this product on Pinte ...	5.0
Baby Tracker® - Daily Childcare Journal, ...	This has been an easy way for my nanny to record ...	4.0

[183531 rows x 3 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.

Build the word count vector for each review

Let us explore a specific example of a baby product.

In [7]:

```
products[269]
```

Out[7]:

```
{'name': 'The First Years Massaging Action Teether',
 'rating': 5.0,
 'review': 'A favorite in our house!'}
```

Now, we will perform 2 simple data transformations:

1. Remove punctuation using Python's built-in (https://docs.python.org/2/library/string.html) string functionality.

2. Transform the reviews into word-counts.

Aside. In this notebook, we remove all punctuations for the sake of simplicity. A smarter approach to punctuations would preserve phrases such as "I'd", "would've", "hadn't" and so forth. See this page (https://www.cis.upenn.edu/~treebank/tokenization.html) for an example of smart handling of punctuations.

In [8]:

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

review_without_punctuation = products['review'].apply(remove_punctuation)
products['word_count'] = graphlab.text_analytics.count_words(review_without_punctuation)
```

Now, let us explore what the sample example above looks like after these 2 transformations. Here, each entry in the **word_count** column is a dictionary where the key is the word and the value is a count of the number of times the word occurs.

In [9]:

```
products[269]['word_count']
```

Out[9]:

```
{'a': 1L, 'favorite': 1L, 'house': 1L, 'in': 1L, 'our': 1L}
```

Extract sentiments

We will **ignore** all reviews with *rating* = *3*, since they tend to have a neutral sentiment.

In [10]:

```
products = products[products['rating'] != 3]
len(products)
```

Out[10]:

```
166752
```

Now, we will assign reviews with a rating of 4 or higher to be *positive* reviews, while the ones with rating of 2 or lower are *negative*. For the sentiment column, we use +1 for the positive class label and -1 for the negative class label.

In [11]:

```
products['sentiment'] = products['rating'].apply(lambda rating : +1 if rating > 3 else -1)
products
```

Out[11]:

name	review	rating	word_count	sentiment
Planetwise Wipe Pouch	it came early and was not disappointed. i love ...	5.0	{'and': 3L, 'love': 1L, 'it': 3L, 'highly': 1L, ...	1
Annas Dream Full Quilt with 2 Shams ...	Very soft and comfortable and warmer than it ...	5.0	{'and': 2L, 'quilt': 1L, 'it': 1L, 'comfortable': ...	1
Stop Pacifier Sucking without tears with ...	This is a product well worth the purchase. I ...	5.0	{'and': 3L, 'ingenious': 1L, 'love': 2L, 'is': ...	1
Stop Pacifier Sucking without tears with ...	All of my kids have cried non-stop when I tried to ...	5.0	{'and': 2L, 'all': 2L, 'help': 1L, 'cried': 1L, ...	1
Stop Pacifier Sucking without tears with ...	When the Binky Fairy came to our house, we didn't ...	5.0	{'and': 2L, 'cute': 1L, 'help': 2L, 'habit': 1L, ...	1
A Tale of Baby's Days with Peter Rabbit ...	Lovely book, it's bound tightly so you may no ...	4.0	{'shop': 1L, 'be': 1L, 'is': 1L, 'bound': 1L, ...	1
Baby Tracker® - Daily Childcare Journal, ...	Perfect for new parents. We were able to keep ...	5.0	{'and': 2L, 'all': 1L, 'right': 1L, 'able': 1L, ...	1
Baby Tracker® - Daily Childcare Journal, ...	A friend of mine pinned this product on Pinte ...	5.0	{'and': 1L, 'fantastic': 1L, 'help': 1L, 'give': ...	1
Baby Tracker® - Daily Childcare Journal, ...	This has been an easy way for my nanny to record ...	4.0	{'all': 1L, 'standarad': 1L, 'another': 1L, ...	1

Baby Tracker® - Daily	I love this journal and our nanny uses it	4.0	{'all': 2L, 'nannys': 1L, 'just': 1L,	1

Now, we can see that the dataset contains an extra column called **sentiment** which is either positive (+1) or negative (-1).

Split data into training and test sets

Let's perform a train/test split with 80% of the data in the training set and 20% of the data in the test set. We use `seed=1` so that everyone gets the same result.

In [12]:

```
train_data, test_data = products.random_split(.8, seed=1)
print len(train_data)
print len(test_data)
```

```
133416
33336
```

Train a sentiment classifier with logistic regression

We will now use logistic regression to create a sentiment classifier on the training data. This model will use the column **word_count** as a feature and the column **sentiment** as the target. We will use `validation_set=None` to obtain same results as everyone else.

Note: This line may take 1-2 minutes.

In [13]:

```
sentiment_model = graphlab.logistic_classifier.create(train_data,
                                                      target = 'sentiment',
                                                      features=['word_count'],
                                                      validation_set=None)
```

```
Logistic regression:

--------------------------------------------------------

Number of examples          : 133416

Number of classes           : 2

Number of feature columns   : 1

Number of unpacked features : 121712

Number of coefficients    : 121713

Starting L-BFGS

--------------------------------------------------------

+-----------+----------+-----------+--------------+-------------------+
| Iteration | Passes   | Step size | Elapsed Time | Training-accuracy |

+-----------+----------+-----------+--------------+-------------------+

| 1         | 5        | 0.000002  | 3.680079     | 0.840754          |

| 2         | 9        | 3.000000  | 5.397031     | 0.931350          |

| 3         | 10       | 3.000000  | 5.921636     | 0.882046          |

| 4         | 11       | 3.000000  | 6.403990     | 0.954076          |

| 5         | 12       | 3.000000  | 6.919377     | 0.960964          |

| 6         | 13       | 3.000000  | 7.370855     | 0.975033          |

+-----------+----------+-----------+--------------+-------------------+

TERMINATED: Terminated due to numerical difficulties.

This model may not be ideal. To improve it, consider doing one of the follow
ing:
(a) Increasing the regularization.
(b) Standardizing the input data.
(c) Removing highly correlated features.
(d) Removing `inf` and `NaN` values in the training data.
```

In [14]:

```
sentiment_model
```

Out[14]:

```
Class                          : LogisticClassifier

Schema
------
Number of coefficients         : 121713
Number of examples             : 133416
Number of classes              : 2
Number of feature columns      : 1
Number of unpacked features    : 121712

Hyperparameters
---------------
L1 penalty                     : 0.0
L2 penalty                     : 0.01

Training Summary
----------------
Solver                         : lbfgs
```

Aside. You may get a warning to the effect of "Terminated due to numerical difficulties --- this model may not be ideal". It means that the quality metric (to be covered in Module 3) failed to improve in the last iteration of the run. The difficulty arises as the sentiment model puts too much weight on extremely rare words. A way to rectify this is to apply regularization, to be covered in Module 4. Regularization lessens the effect of extremely rare words. For the purpose of this assignment, however, please proceed with the model above.

Now that we have fitted the model, we can extract the weights (coefficients) as an SFrame as follows:

In [15]:

```
weights = sentiment_model.coefficients
weights.column_names()
```

Out[15]:

```
['name', 'index', 'class', 'value', 'stderr']
```

There are a total of `121713` coefficients in the model. Recall from the lecture that positive weights w_j correspond to weights that cause positive sentiment, while negative weights correspond to negative sentiment.

Fill in the following block of code to calculate how many *weights* are positive (>= 0). (**Hint**: The `'value'` column in SFrame *weights* must be positive (>= 0)).

In [17]:

```
num_positive_weights = (weights['value'] >= 0).sum()
num_negative_weights = (weights['value'] < 0).sum()

print "Number of positive weights: %s " % num_positive_weights
print "Number of negative weights: %s " % num_negative_weights
```

```
Number of positive weights: 68419
Number of negative weights: 53294
```

Quiz Question: How many weights are >= 0?

Making predictions with logistic regression

Now that a model is trained, we can make predictions on the **test data**. In this section, we will explore this in the context of 3 examples in the test dataset. We refer to this set of 3 examples as the **sample_test_data**.

In [18]:

```
sample_test_data = test_data[10:13]
print sample_test_data['rating']
sample_test_data
```

```
[5.0, 2.0, 1.0]
```

Out[18]:

name	review	rating	word_count	sentiment
Our Baby Girl Memory Book	Absolutely love it and all of the Scripture in ...	5.0	{'and': 2L, 'all': 1L, 'love': 1L, ...	1
Wall Decor Removable Decal Sticker - Colorful ...	Would not purchase again or recommend. The decals ...	2.0	{'and': 1L, 'wall': 1L, 'them': 1L, 'decals': ...	-1
New Style Trailing Cherry Blossom Tree Decal ...	Was so excited to get this product for my baby ...	1.0	{'all': 1L, 'money': 1L, 'into': 1L, 'it': 3L, ...	-1

[3 rows x 5 columns]

Let's dig deeper into the first row of the **sample_test_data**. Here's the full review:

In [19]:

```
sample_test_data[0]['review']
```

Out[19]:

```
'Absolutely love it and all of the Scripture in it.  I purchased the Baby Bo
y version for my grandson when he was born and my daughter-in-law was thrill
ed to receive the same book again.'
```

That review seems pretty positive.

Now, let's see what the next row of the **sample_test_data** looks like. As we could guess from the sentiment (-1), the review is quite negative.

In [20]:

```
sample_test_data[1]['review']
```

Out[20]:

```
'Would not purchase again or recommend. The decals were thick almost plastic
like and were coming off the wall as I was applying them! The would NOT stic
k! Literally stayed stuck for about 5 minutes then started peeling off.'
```

We will now make a **class** prediction for the **sample_test_data**. The `sentiment_model` should predict **+1** if the sentiment is positive and **-1** if the sentiment is negative. Recall from the lecture that the **score** (sometimes called **margin**) for the logistic regression model is defined as:

$$\text{score}_i = \mathbf{w}^T h(\mathbf{x}_i)$$

where $h(\mathbf{x}_i)$ represents the features for example i. We will write some code to obtain the **scores** using GraphLab Create. For each row, the **score** (or margin) is a number in the range **[-inf, inf]**.

In [21]:

```
scores = sentiment_model.predict(sample_test_data, output_type='margin')
print scores
```

```
[6.734619727060332, -5.734130996760911, -14.668460404469558]
```

Predicting sentiment

These scores can be used to make class predictions as follows:

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

Using scores, write code to calculate $\hat{y}$, the class predictions:

In [22]:

```
def class_predictions(scores):
    preds = []
    for score in scores:
        if score > 0:
            pred = 1
        else:
            pred = -1
        preds.append(pred)
    return preds
```

In [23]:

```
class_predictions(scores)
```

Out[23]:

```
[1, -1, -1]
```

Run the following code to verify that the class predictions obtained by your calculations are the same as that obtained from GraphLab Create.

In [24]:

```
print "Class predictions according to GraphLab Create:"
print sentiment_model.predict(sample_test_data)
```

```
Class predictions according to GraphLab Create:
[1L, -1L, -1L]
```

Checkpoint: Make sure your class predictions match with the one obtained from GraphLab Create.

Probability predictions

Recall from the lectures that we can also calculate the probability predictions from the scores using:

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

Using the variable **scores** calculated previously, write code to calculate the probability that a sentiment is positive using the above formula. For each row, the probabilities should be a number in the range **[0, 1]**.

In [25]:

```
def conditional_prob(scores):
    prob_preds = []
    for score in scores:
        prob_pred = 1 / (1 + math.exp(-score))
        prob_preds.append(prob_pred)
    return prob_preds

conditional_prob(scores)
```

Out[25]:

```
[0.998812384837721, 0.003223268181798583, 4.2615579966509917e-07]
```

Checkpoint: Make sure your probability predictions match the ones obtained from GraphLab Create.

In [26]:

```
print "Class predictions according to GraphLab Create:"
print sentiment_model.predict(sample_test_data, output_type='probability')
```

```
Class predictions according to GraphLab Create:
[0.998812384837721, 0.003223268181798584, 4.2615579966509896e-07]
```

In [28]:

```
def pred_output(data):
    for d in data:
        if d > 0.5:
            print 1
        else:
            print -1

pred_output(conditional_prob(scores))
```

```
1
-1
-1
```

Quiz Question: Of the three data points in **sample_test_data**, which one (first, second, or third) has the **lowest probability** of being classified as a positive review?

Find the most positive (and negative) review

We now turn to examining the full test dataset, **test_data**, and use GraphLab Create to form predictions on all of the test data points for faster performance.

Using the `sentiment_model`, find the 20 reviews in the entire **test_data** with the **highest probability** of being classified as a **positive review**. We refer to these as the "most positive reviews."

To calculate these top-20 reviews, use the following steps:

1. Make probability predictions on **test_data** using the `sentiment_model`. (**Hint:** When you call `.predict` to make predictions on the test data, use option `output_type='probability'` to output the probability rather than just the most likely class.)
2. Sort the data according to those predictions and pick the top 20. (**Hint:** You can use the `.topk` method on an SFrame to find the top k rows sorted according to the value of a specified column.)

In [29]:

```
test_data['proba_pred'] = sentiment_model.predict(test_data, output_type='probability')
test_data
```

Out[29]:

name	review	rating	word_count	sentiment
Baby Tracker® - Daily Childcare Journal, ...	This has been an easy way for my nanny to record ...	4.0	{'all': 1L, 'standarad': 1L, 'another': 1L, ...	1
Baby Tracker® - Daily Childcare Journal, ...	I love this journal and our nanny uses it ...	4.0	{'all': 2L, 'nannys': 1L, 'just': 1L, 'sleep': 2L, ...	1
Nature's Lullabies First Year Sticker Calendar ...	I love this little calender, you can keep ...	5.0	{'and': 1L, 'babys': 1L, 'love': 1L, 'like': 1L, ...	1
Nature's Lullabies Second Year Sticker Calendar ...	I had a hard time finding a second year calendar, ...	5.0	{'and': 3L, 'all': 1L, 'months': 1L, ...	1
Lamaze Peekaboo, I Love You ...	One of baby's first and favorite books, and i ...	4.0	{'and': 2L, 'because': 1L, 'family': 1L, ...	1
Lamaze Peekaboo, I Love You ...	My son loved this book as an infant. It was ...	5.0	{'all': 1L, 'being': 1L, 'infant': 1L, 'course': ...	1
Lamaze Peekaboo, I Love You ...	Our baby loves this book & has loved it for a ...	5.0	{'and': 1L, 'own': 1L, 'it': 3L, 'our': 1L, ...	1
SoftPlay Giggle Jiggle Funbook, Happy Bear ...	This bear is absolutely adorable and I would ...	2.0	{'and': 3L, 'cute': 1L, 'rating': 1L, ...	-1
SoftPlay Peek-A-Boo Where's Elmo A Childr ...	I bought two for recent baby showers! The book ...	5.0	{'beautiful': 1L, 'and': 2L, 'love': 1L, 'elmo': ...	1

Baby's First Year Undated Wall Calendar	I searched high and low for a first year	5.0	{'remembering': 1L, 'and': 4L, ...	1

In [56]:

```
test_data['name','proba_pred'].topk('proba_pred', k=20).print_rows(20)
```

```
+------------------------------+------------+
|              name            | proba_pred |
+------------------------------+------------+
| Britax Decathlon Convertib... |    1.0     |
| bumGenius One-Size Cloth D... |    1.0     |
| Moby Wrap Original 100% Co... |    1.0     |
| Moby Wrap Original 100% Co... |    1.0     |
| Ameda Purely Yours Breast ... |    1.0     |
| Traveling Toddler Car Seat... |    1.0     |
| Cloud b Sound Machine Soot... |    1.0     |
| JP Lizzy Chocolate Ice Cla... |    1.0     |
| Lilly Gold Sit 'n' Stroll ... |    1.0     |
|  Fisher-Price Deluxe Jumperoo |    1.0     |
|   Munchkin Mozart Magic Cube  |    1.0     |
| Britax Marathon Convertibl... |    1.0     |
| Wizard Convertible Car Sea... |    1.0     |
|   Capri Stroller - Red Tech   |    1.0     |
| Peg Perego Primo Viaggio C... |    1.0     |
| HALO SleepSack Micro-Fleec... |    1.0     |
| Leachco Snoogle Total Body... |    1.0     |
| Summer Infant Complete Nur... |    1.0     |
| Safety 1st Tot-Lok Four Lo... |    1.0     |
|  BABYBJORN Potty Chair - Red  |    1.0     |
+------------------------------+------------+
[20 rows x 2 columns]
```

Quiz Question: Which of the following products are represented in the 20 most positive reviews? [multiple choice]

Now, let us repeat this exercise to find the "most negative reviews." Use the prediction probabilities to find the 20 reviews in the **test_data** with the **lowest probability** of being classified as a **positive review**. Repeat the same steps above but make sure you **sort in the opposite order**.

In [31]:

```
test_data['name','proba_pred'].topk('proba_pred', k=20, reverse=True).print_rows(20)
```

```
+------------------------------+--------------------+
|              name            |     proba_pred     |
+------------------------------+--------------------+
| Jolly Jumper Arctic Sneak ... | 7.80415068204e-100 |
| Levana Safe N'See Digital ... |  6.8365088551e-25  |
| Snuza Portable Baby Moveme... | 2.12654510822e-24  |
| Fisher-Price Ocean Wonders... | 2.24582080778e-23  |
| VTech Communications Safe ... | 1.32962966148e-22  |
| Safety 1st High-Def Digita... | 2.06872097469e-20  |
| Chicco Cortina KeyFit 30 T... | 5.93881994668e-20  |
| Prince Lionheart Warmies W... | 6.28510016533e-20  |
| Valco Baby Tri-mode Twin S... | 8.05528712682e-20  |
| Adiri BPA Free Natural Nur... |  8.4652172493e-20  |
| Munchkin Nursery Projector... | 1.52853945169e-19  |
| The First Years True Choic... | 1.77901889389e-19  |
| Nuby Natural Touch Silicon... | 1.15227353847e-18  |
| Peg-Perego Tatamia High Ch... | 1.26175666135e-18  |
|    Fisher-Price Royal Potty   | 1.60282966314e-18  |
| Safety 1st Exchangeable Ti... | 7.04887411708e-18  |
| Safety 1st Lift Lock and S... | 9.84839237567e-18  |
| Evenflo Take Me Too Premie... | 1.00120730395e-17  |
| Cloth Diaper Sprayer--styl... |  1.169063556e-17   |
| The First Years 3 Pack Bre... | 1.22003532002e-17  |
+------------------------------+--------------------+
[20 rows x 2 columns]
```

Quiz Question: Which of the following products are represented in the 20 most negative reviews? [multiple choice]

Compute accuracy of the classifier

We will now evaluate the accuracy of the trained classifier. Recall that the accuracy is given by

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

This can be computed as follows:

- **Step 1:** Use the trained model to compute class predictions (**Hint:** Use the `predict` method)
- **Step 2:** Count the number of data points when the predicted class labels match the ground truth labels (called `true_labels` below).
- **Step 3:** Divide the total number of correct predictions by the total number of data points in the dataset.

Complete the function below to compute the classification accuracy:

In [32]:

```
def get_classification_accuracy(model, data, true_labels):
    # First get the predictions
    predictions = model.predict(data)

    # Compute the number of correctly classified examples
    num_correct = sum(predictions == true_labels)

    # Then compute accuracy by dividing num_correct by total number of examples
    accuracy = num_correct / len(data)

    return accuracy
```

Now, let's compute the classification accuracy of the **sentiment_model** on the **test_data**.

In [33]:

```
get_classification_accuracy(sentiment_model, test_data, test_data['sentiment'])
```

Out[33]:

```
0.9145368370530358
```

In [35]:

```
get_classification_accuracy(sentiment_model, train_data,
                            train_data['sentiment'])
```

Out[35]:

```
0.979440247046831
```

Quiz Question: What is the accuracy of the **sentiment_model** on the **test_data**? Round your answer to 2 decimal places (e.g. 0.76).

Quiz Question: Does a higher accuracy value on the **training_data** always imply that the classifier is better?

Learn another classifier with fewer words

There were a lot of words in the model we trained above. We will now train a simpler logistic regression model using only a subset of words that occur in the reviews. For this assignment, we selected a 20 words to work with. These are:

In [36]:

```
significant_words = ['love', 'great', 'easy', 'old', 'little', 'perfect', 'loves',
      'well', 'able', 'car', 'broke', 'less', 'even', 'waste', 'disappointed',
      'work', 'product', 'money', 'would', 'return']
```

In [37]:

```
len(significant_words)
```

Out[37]:

```
20
```

For each review, we will use the **word_count** column and trim out all words that are **not** in the

significant_words list above. We will use the SArray dictionary trim by keys functionality (https://dato.com/products/create/docs/generated/graphlab.SArray.dict_trim_by_keys.html). Note that we are performing this on both the training and test set.

In [38]:

```
train_data['word_count_subset'] = train_data['word_count'].dict_trim_by_keys(significant_wo
test_data['word_count_subset'] = test_data['word_count'].dict_trim_by_keys(significant_word
```

Let's see what the first example of the dataset looks like:

In [39]:

```
train_data[0]['review']
```

Out[39]:

```
'it came early and was not disappointed. i love planet wise bags and now my
wipe holder. it keps my osocozy wipes moist and does not leak. highly recomm
end it.'
```

The **word_count** column had been working with before looks like the following:

In [40]:

```
print train_data[0]['word_count']
```

```
{'and': 3L, 'love': 1L, 'it': 3L, 'highly': 1L, 'osocozy': 1L, 'bags': 1L,
'leak': 1L, 'moist': 1L, 'does': 1L, 'recommend': 1L, 'was': 1L, 'wipes': 1
L, 'disappointed': 1L, 'early': 1L, 'not': 2L, 'now': 1L, 'holder': 1L, 'wip
e': 1L, 'keps': 1L, 'wise': 1L, 'i': 1L, 'planet': 1L, 'my': 2L, 'came': 1L}
```

Since we are only working with a subset of these words, the column **word_count_subset** is a subset of the above dictionary. In this example, only 2 `significant words` are present in this review.

In [41]:

```
print train_data[0]['word_count_subset']
```

```
{'love': 1L, 'disappointed': 1L}
```

Train a logistic regression model on a subset of data

We will now build a classifier with **word_count_subset** as the feature and **sentiment** as the target.

In [42]:

```
simple_model = graphlab.logistic_classifier.create(train_data,
                                                   target = 'sentiment',
                                                   features=['word_count_subset'],
                                                   validation_set=None)
simple_model
```

```
Logistic regression:

--------------------------------------------------------

Number of examples          : 133416

Number of classes           : 2

Number of feature columns   : 1

Number of unpacked features : 20

Number of coefficients    : 21

Starting Newton Method

--------------------------------------------------------

+-----------+----------+--------------+-------------------+

| Iteration | Passes   | Elapsed Time | Training-accuracy |

+-----------+----------+--------------+-------------------+

| 1         | 2        | 0.240449     | 0.862917          |

| 2         | 3        | 0.400105     | 0.865713          |

| 3         | 4        | 0.518621     | 0.866478          |

| 4         | 5        | 0.626591     | 0.866748          |

| 5         | 6        | 0.778571     | 0.866815          |

| 6         | 7        | 0.890543     | 0.866815          |

+-----------+----------+--------------+-------------------+

SUCCESS: Optimal solution found.
```

Out[42]:

```
Class                         : LogisticClassifier

Schema
------
Number of coefficients        : 21
Number of examples            : 133416
Number of classes             : 2
Number of feature columns     : 1
Number of unpacked features   : 20

Hyperparameters
---------------
L1 penalty                    : 0.0
L2 penalty                    : 0.01
```

```
Training Summary
----------------
Solver                        : newton
Solver iterations             : 6
Solver status                 : SUCCESS: Optimal solution found.
Training time (sec)           : 0.9185

Settings
--------
Log-likelihood                : 44323.7254

Highest Positive Coefficients
-----------------------------
word_count_subset[loves]      : 1.6773
word_count_subset[perfect]    : 1.5145
word_count_subset[love]       : 1.3654
(intercept)                   : 1.2995
word_count_subset[easy]       : 1.1937

Lowest Negative Coefficients
----------------------------
word_count_subset[disappointed] : -2.3551
word_count_subset[return]     : -2.1173
word_count_subset[waste]      : -2.0428
word_count_subset[broke]      : -1.658
word_count_subset[money]      : -0.8979
```

We can compute the classification accuracy using the `get_classification_accuracy` function you implemented earlier.

In [43]:

```
get_classification_accuracy(simple_model, test_data, test_data['sentiment'])
```

Out[43]:

```
0.8693004559635229
```

Now, we will inspect the weights (coefficients) of the **simple_model**:

In [44]:

```
simple_model.coefficients
```

Out[44]:

name	index	class	value	stderr
(intercept)	None	1	1.2995449552	0.0120888541331
word_count_subset	disappointed	1	-2.35509250061	0.0504149888557
word_count_subset	love	1	1.36543549368	0.0303546295109
word_count_subset	well	1	0.504256746398	0.021381300631
word_count_subset	product	1	-0.320555492996	0.0154311321362
word_count_subset	loves	1	1.67727145556	0.0482328275384
word_count_subset	little	1	0.520628636025	0.0214691475665
word_count_subset	work	1	-0.621700012425	0.0230330597946
word_count_subset	easy	1	1.19366189833	0.029288869202
word_count_subset	great	1	0.94469126948	0.0209509926591

[21 rows x 5 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.

Let's sort the coefficients (in descending order) by the **value** to obtain the coefficients with the most positive effect on the sentiment.

In [45]:

```
simple_model.coefficients.sort('value', ascending=False).print_rows(num_rows=21)
```

```
+------------------+-------------+-------+----------------+-------------
---+
|        name      |    index    | class |      value     |      stderr
   |
+------------------+-------------+-------+----------------+-------------
---+
| word_count_subset |    loves    |   1   |  1.67727145556  | 0.04823282753
84 |
| word_count_subset |   perfect   |   1   |  1.51448626703  |  0.0498619522
94 |
| word_count_subset |     love    |   1   |  1.36543549368  | 0.03035462951
09 |
|    (intercept)    |     None    |   1   |   1.2995449552  | 0.01208885413
31 |
| word_count_subset |     easy    |   1   |  1.19366189833  |  0.0292888692
02 |
| word_count_subset |    great    |   1   |  0.94469126948  | 0.02095099265
91 |
| word_count_subset |    little   |   1   |  0.520628636025 | 0.02146914756
65 |
| word_count_subset |     well    |   1   |  0.504256746398 |  0.0213813006
31 |
| word_count_subset |     able    |   1   |  0.191438302295 | 0.03375819556
97 |
| word_count_subset |     old     |   1   | 0.0853961886678 | 0.02008634230
25 |
| word_count_subset |     car     |   1   |  0.058834990068 | 0.01682915320
91 |
| word_count_subset |     less    |   1   | -0.209709815216 |  0.0405057359
54 |
| word_count_subset |   product   |   1   | -0.320555492996 | 0.01543113213
62 |
| word_count_subset |    would    |   1   | -0.362308947711 | 0.01275447519
85 |
| word_count_subset |     even    |   1   |  -0.51173855127 | 0.01996127602
61 |
| word_count_subset |     work    |   1   | -0.621700012425 | 0.02303305979
46 |
| word_count_subset |    money    |   1   | -0.897884155776 | 0.03399367328
36 |
| word_count_subset |    broke    |   1   |  -1.65796447838 | 0.05808789071
66 |
| word_count_subset |    waste    |   1   |   -2.042773611  | 0.06447029324
44 |
| word_count_subset |    return   |   1   |  -2.11729659718 | 0.05786508072
41 |
| word_count_subset | disappointed |   1   |  -2.35509250061 | 0.05041498885
57 |
+------------------+-------------+-------+----------------+-------------
---+
[21 rows x 5 columns]
```

Quiz Question: Consider the coefficients of **simple_model**. There should be 21 of them, an intercept term + one for each word in **significant_words**. How many of the 20 coefficients (corresponding to the 20 **significant_words** and *excluding the intercept term*) are positive for the `simple_model`?

In [46]:

```
simple_weights = simple_model.coefficients
positive_significant_words = simple_weights[(simple_weights['value'] > 0) & (simple_weights
print len(positive_significant_words)
print positive_significant_words
```

```
10
['love', 'well', 'loves', 'little', 'easy', 'great', 'able', 'perfect', 'ol
d', 'car']
```

Quiz Question: Are the positive words in the **simple_model** (let us call them `positive_significant_words`) also positive words in the **sentiment_model**?

In [47]:

```
weights.filter_by(positive_significant_words, 'index')
```

Out[47]:

name	index	class	value	stderr
word_count	love	1	1.43301685439	None
word_count	well	1	0.627964877567	None
word_count	loves	1	1.5664851757	None
word_count	little	1	0.674162457499	None
word_count	easy	1	1.21346937822	None
word_count	great	1	1.31459245039	None
word_count	able	1	0.174331272552	None
word_count	perfect	1	1.75190114392	None
word_count	old	1	0.00912230113667	None
word_count	car	1	0.195263670618	None

[10 rows x 5 columns]

Comparing models

We will now compare the accuracy of the **sentiment_model** and the **simple_model** using the `get_classification_accuracy` method you implemented above.

First, compute the classification accuracy of the **sentiment_model** on the **train_data**:

In [48]:

```
get_classification_accuracy(sentiment_model, train_data, train_data['sentiment'])
```

Out[48]:

```
0.979440247046831
```

Now, compute the classification accuracy of the **simple_model** on the **train_data**:

In [49]:

```
get_classification_accuracy(simple_model, train_data, train_data['sentiment'])
```

Out[49]:

```
0.8668150746537147
```

Quiz Question: Which model (**sentiment_model** or **simple_model**) has higher accuracy on the TRAINING set?

Now, we will repeat this exercise on the **test_data**. Start by computing the classification accuracy of the **sentiment_model** on the **test_data**:

In [50]:

```
get_classification_accuracy(sentiment_model, test_data, test_data['sentiment'])
```

Out[50]:

```
0.9145368370530358
```

Next, we will compute the classification accuracy of the **simple_model** on the **test_data**:

In [51]:

```
get_classification_accuracy(simple_model, test_data, test_data['sentiment'])
```

Out[51]:

```
0.8693004559635229
```

Quiz Question: Which model (**sentiment_model** or **simple_model**) has higher accuracy on the TEST set?

Baseline: Majority class prediction

It is quite common to use the **majority class classifier** as the a baseline (or reference) model for comparison with your classifier model. The majority classifier model predicts the majority class for all data points. At the very least, you should healthily beat the majority class classifier, otherwise, the model is (usually) pointless.

What is the majority class in the **train_data**?

```
---
A majority class classifier is "trained" to guess the majority class of the traini
ng set. For example, if the majority of reviews in the training set are positive,
 then the majority class will always guess that a review is positive.
```

In [52]:

```
num_positive  = (train_data['sentiment'] == +1).sum()
num_negative = (train_data['sentiment'] == -1).sum()
print num_positive
print num_negative
```

```
112164
21252
```

Now compute the accuracy of the majority class classifier on **test_data**.

Quiz Question: Enter the accuracy of the majority class classifier model on the **test_data**. Round your answer to two decimal places (e.g. 0.76).

In [53]:

```
num_positive_test  = (test_data['sentiment'] == +1).sum()
num_negative_test = (test_data['sentiment'] == -1).sum()
print num_positive_test
print num_negative_test

accuracy_test = num_positive_test / len(test_data['sentiment'])
print accuracy_test
```

```
28095
5241
0.842782577394
```

Quiz Question: Is the **sentiment_model** definitely better than the majority class classifier (the baseline)?

Quiz

1 point

1.
How many weights are greater than or equal to 0?

68419

1 point

2.
Of the three data points in sample_test_data, which one has the lowest probability of being classified as a positive review?

- First
- Second
- Third

3.
Which of the following products are represented in the 20 most positive reviews?

- [ ] Snuza Portable Baby Movement Monitor
- [ ] MamaDoo Kids Foldable Play Yard Mattress Topper, Blue
- [x] Britax Decathlon Convertible Car Seat, Tiffany
- [ ] Safety 1st Exchangeable Tip 3 in 1 Thermometer

1 point

4.
Which of the following products are represented in the 20 most negative reviews?

- [x] The First Years True Choice P400 Premium Digital Monitor, 2 Parent Unit
- [ ] JP Lizzy Chocolate Ice Classic Tote Set
- [x] Peg-Perego Tatamia High Chair, White Latte
- [x] Safety 1st High-Def Digital Monitor

5.
What is the accuracy of the sentiment_model on the test_data? Round your answer to 2 decimal places (e.g. 0.76).

0.91

1 point

6.
Does a higher accuracy value on the training_data always imply that the classifier is better?

- ( ) Yes, higher accuracy on training data always implies that the classifier is better.
- (•) No, higher accuracy on training data does not necessarily imply that the classifier is better.

7.
Consider the coefficients of simple_model. There should be 21 of them, an intercept term + one for each word in significant_words.

How many of the 20 coefficients (corresponding to the 20 significant_words and excluding the intercept term) are positive for the simple_model?

10

1 point

8.
Are the positive words in the simple_model also positive words in the sentiment_model?

- (●) Yes
- ( ) No

9.
Which model (sentiment_model or simple_model) has higher accuracy on the TRAINING set?

- (●) Sentiment_model
- ( ) Simple_model

1 point

10.
Which model (sentiment_model or simple_model) has higher accuracy on the TEST set?

- (●) Sentiment_model
- ( ) Simple_model

11.
Enter the accuracy of the majority class classifier model on the test_data. Round your answer to two decimal places (e.g. 0.76).

0.84

1 point

12.
Is the sentiment_model definitely better than the majority class classifier (the baseline)?

- (●) Yes
- ( ) No