Exploring precision and recall

The goal of this second notebook is to understand precision-recall in the context of classifiers.

- Use Amazon review data in its entirety.
- Train a logistic regression model.
- Explore various evaluation metrics: accuracy, confusion matrix, precision, recall.
- Explore how various metrics can be combined to produce a cost of making an error.
- Explore precision and recall curves.

Because we are using the full Amazon review dataset (not a subset of words or reviews), in this assignment we return to using GraphLab Create for its efficiency. As usual, let's start by **firing up GraphLab Create**.

Make sure you have the latest version of GraphLab Create (1.8.3 or later). If you don't find the decision tree module, then you would need to upgrade graphlab-create using

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

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

In [1]:

```python
import graphlab
from __future__ import division
import numpy as np
graphlab.canvas.set_target('ipynb')
```

Load amazon review dataset

In [2]:

```python
products = graphlab.SFrame('amazon_baby.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_1534003800.log.0
```

Extract word counts and sentiments

As in the first assignment of this course, we compute the word counts for individual words and extract positive and negative sentiments from ratings. To summarize, we perform the following:

1. Remove punctuation.
2. Remove reviews with "neutral" sentiment (rating 3).
3. Set reviews with rating 4 or more to be positive and those with 2 or less to be negative.

In [3]:

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

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

# Count words
products['word_count'] = graphlab.text_analytics.count_words(review_clean)

# Drop neutral sentiment reviews.
products = products[products['rating'] != 3]

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

Now, let's remember what the dataset looks like by taking a quick peek:

In [4]:

```
products
```

Out[4]:

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&reg; - 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&reg; - Daily Childcare Journal, ...	A friend of mine pinned this product on Pinte ...	5.0	{'and': 1L, 'fantastic': 1L, 'help': 1L, 'give': ...	1
Baby Tracker&reg; - 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&reg; -	I love this journal and	4.0	{'all': 2L, 'nannys': 1L,	1

Split data into training and test sets

We split the data into a 80-20 split where 80% is in the training set and 20% is in the test set.

In [5]:

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

Train a logistic regression classifier

We will now train a logistic regression classifier with **sentiment** as the target and **word_count** as the features. We will set `validation_set=None` to make sure everyone gets exactly the same results.

Remember, even though we now know how to implement logistic regression, we will use GraphLab Create for its efficiency at processing this Amazon dataset in its entirety. The focus of this assignment is instead on the topic of precision and recall.

In [6]:

```
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 |
```

Model Evaluation

We will explore the advanced model evaluation concepts that were discussed in the lectures.

Accuracy

One performance metric we will use for our more advanced exploration is accuracy, which we have seen many times in past assignments. Recall that the accuracy is given by

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

To obtain the accuracy of our trained models using GraphLab Create, simply pass the option `metric='accuracy'` to the `evaluate` function. We compute the **accuracy** of our logistic regression model on the **test_data** as follows:

In [7]:

```
accuracy= model.evaluate(test_data, metric='accuracy')['accuracy']
print "Test Accuracy: %s" % accuracy
```

```
Test Accuracy: 0.914536837053
```

Baseline: Majority class prediction

Recall from an earlier assignment that we used the **majority class classifier** as a baseline (i.e reference) model for a point of comparison with a more sophisticated classifier. The majority classifier model predicts the majority class for all data points.

Typically, a good model should beat the majority class classifier. Since the majority class in this dataset is the positive class (i.e., there are more positive than negative reviews), the accuracy of the majority class classifier can be computed as follows:

In [8]:

```
baseline = len(test_data[test_data['sentiment'] == 1])/len(test_data)
print "Baseline accuracy (majority class classifier): %s" % baseline
```

```
Baseline accuracy (majority class classifier): 0.842782577394
```

Quiz Question: Using accuracy as the evaluation metric, was our **logistic regression model** better than the baseline (majority class classifier)?

- Baseline accuracy (majority class classifier): 0.842782577394
- Test Accuracy (logistic regression): 0.914536837053
- yes, logistic regression has higher accuracy than majority class classifier

Confusion Matrix

The accuracy, while convenient, does not tell the whole story. For a fuller picture, we turn to the **confusion matrix**. In the case of binary classification, the confusion matrix is a 2-by-2 matrix laying out correct and incorrect predictions made in each label as follows:

```
              +---------------------------------------------+
              |                Predicted label              |
              +----------------------+----------------------+
              |          (+1)        |         (-1)         |
+-------+-----+----------------------+----------------------+
| True  |(+1) | # of true positives  | # of false negatives |
| label +-----+----------------------+----------------------+
|       |(-1) | # of false positives | # of true negatives  |
+-------+-----+----------------------+----------------------+
```

To print out the confusion matrix for a classifier, use `metric='confusion_matrix'`:

In [9]:

```
confusion_matrix = model.evaluate(test_data, metric='confusion_matrix')['confusion_matrix']
confusion_matrix
```

Out[9]:

target_label	predicted_label	count
-1	-1	3798
-1	1	1443
1	-1	1406
1	1	26689

[4 rows x 3 columns]

Quiz Question: How many predicted values in the **test set** are **false positives**?

They are negatives - predicted as positives -- 1443

Computing the cost of mistakes

Put yourself in the shoes of a manufacturer that sells a baby product on Amazon.com and you want to monitor your product's reviews in order to respond to complaints. Even a few negative reviews may generate a lot of bad publicity about the product. So you don't want to miss any reviews with negative sentiments --- you'd rather put up with false alarms about potentially negative reviews instead of missing negative reviews entirely. In other words, **false positives cost more than false negatives**. (It may be the other way around for other scenarios, but let's stick with the manufacturer's scenario for now.)

Suppose you know the costs involved in each kind of mistake:

1. $100 for each false positive.
2. $1 for each false negative.
3. Correctly classified reviews incur no cost.

Quiz Question: Given the stipulation, what is the cost associated with the logistic regression classifier's performance on the **test set**?

In [12]:

```
cost = 1443 * 100 + 1406 * 1
cost
```

Out[12]:

```
145706
```

In [13]:

```
list(confusion_matrix['count'])[1] * 100 + list(confusion_matrix['count'])[2] * 1
```

Out[13]:

```
145706L
```

Precision and Recall

You may not have exact dollar amounts for each kind of mistake. Instead, you may simply prefer to reduce the percentage of false positives to be less than, say, 3.5% of all positive predictions. This is where **precision** comes in:

$$\begin{aligned}[\text{precision}] &= \frac{[\text{\# positive data points with positive predicitions}]}{[\text{\# all data points with positive predictions}]} \\ &= \frac{[\text{\# true positives}]}{[\text{\# true positives}] + [\text{\# false positives}]}\end{aligned}$$

So to keep the percentage of false positives below 3.5% of positive predictions, we must raise the precision to 96.5% or higher.

First, let us compute the precision of the logistic regression classifier on the **test_data**.

In [14]:

```
precision = model.evaluate(test_data, metric='precision')['precision']
print "Precision on test data: %s" % precision
```

```
Precision on test data: 0.948706099815
```

Quiz Question: Out of all reviews in the **test set** that are predicted to be positive, what fraction of them are **false positives**? (Round to the second decimal place e.g. 0.25)

In [18]:

```
f_pos = 1443 / (26689+1443)
f_pos
```

Out[18]:

```
0.05129390018484288
```

In [19]:

```
false_positive = confusion_matrix['count'][(confusion_matrix['target_label'] == -1) & (conf
false_positive

false_positive/confusion_matrix['count'].sum()
```

Out[19]:

```
dtype: float
Rows: ?
[0.04328653707703384, ... ]
```

Quiz Question: Based on what we learned in lecture, if we wanted to reduce this fraction of false positives to be below 3.5%, we would: (see the quiz)

Increase threshold for predicting the positive class (y^=+1)

A complementary metric is **recall**, which measures the ratio between the number of true positives and that of (ground-truth) positive reviews:

$$\begin{aligned}[\text{recall}] &= \frac{[\text{\# positive data points with positive predicitions}]}{[\text{\# all positive data points}]} \\ &= \frac{[\text{\# true positives}]}{[\text{\# true positives}] + [\text{\# false negatives}]}\end{aligned}$$

Let us compute the recall on the **test_data**.

In [20]:

```
recall = model.evaluate(test_data, metric='recall')['recall']
print "Recall on test data: %s" % recall
```

```
Recall on test data: 0.949955508098
```

Quiz Question: What fraction of the positive reviews in the **test_set** were correctly predicted as positive by the classifier?

In [21]:

```
print round(recall, 3) , " of the positive reviews in the test_set were correctly predictec
```

```
0.95  of the positive reviews in the test_set were correctly predicted as po
sitive by the classifier
```

Quiz Question: What is the recall value for a classifier that predicts **+1** for all data points in the **test_data**?

The recall value for a classifier that predicts +1 for all data points in the test_data is 1.0

Precision-recall tradeoff

In this part, we will explore the trade-off between precision and recall discussed in the lecture. We first examine what happens when we use a different threshold value for making class predictions. We then explore a range of threshold values and plot the associated precision-recall curve.

Varying the threshold

False positives are costly in our example, so we may want to be more conservative about making positive predictions. To achieve this, instead of thresholding class probabilities at 0.5, we can choose a higher threshold.

Write a function called `apply_threshold` that accepts two things

- `probabilities` (an SArray of probability values)
- `threshold` (a float between 0 and 1).

The function should return an SArray, where each element is set to +1 or -1 depending whether the corresponding probability exceeds `threshold`.

In [23]:

```
def apply_threshold(probabilities, threshold):
    ### YOUR CODE GOES HERE
    # +1 if >= threshold and -1 otherwise.
    answer = graphlab.SArray([+1 if x >= threshold else -1 for x in probabilities])
    return answer
```

Run prediction with `output_type='probability'` to get the list of probability values. Then use thresholds set at 0.5 (default) and 0.9 to make predictions from these probability values.

In [24]:

```
probabilities = model.predict(test_data, output_type='probability')
predictions_with_default_threshold = apply_threshold(probabilities, 0.5)
predictions_with_high_threshold = apply_threshold(probabilities, 0.9)
```

In [25]:

```
print "Number of positive predicted reviews (threshold = 0.5): %s" % (predictions_with_defa
```

```
Number of positive predicted reviews (threshold = 0.5): 28132
```

In [26]:

```
print "Number of positive predicted reviews (threshold = 0.9): %s" % (predictions_with_high
```

```
Number of positive predicted reviews (threshold = 0.9): 25630
```

Quiz Question: What happens to the number of positive predicted reviews as the threshold increased from 0.5 to 0.9?

- Number of positive predicted reviews (threshold = 0.5): 28132
- Number of positive predicted reviews (threshold = 0.9): 25630
- Number of positive predicted reviews reduced as the threshold increased from 0.5 to 0.9.

Exploring the associated precision and recall as the threshold varies

By changing the probability threshold, it is possible to influence precision and recall. We can explore this as follows:

In [27]:

```
# Threshold = 0.5
precision_with_default_threshold = graphlab.evaluation.precision(test_data['sentiment'],
                                        predictions_with_default_threshold)

recall_with_default_threshold = graphlab.evaluation.recall(test_data['sentiment'],
                                        predictions_with_default_threshold)

# Threshold = 0.9
precision_with_high_threshold = graphlab.evaluation.precision(test_data['sentiment'],
                                        predictions_with_high_threshold)
recall_with_high_threshold = graphlab.evaluation.recall(test_data['sentiment'],
                                        predictions_with_high_threshold)
```

In [28]:

```
print "Precision (threshold = 0.5): %s" % precision_with_default_threshold
print "Recall (threshold = 0.5)   : %s" % recall_with_default_threshold
```

```
Precision (threshold = 0.5): 0.948706099815
Recall (threshold = 0.5)   : 0.949955508098
```

In [29]:

```
print "Precision (threshold = 0.9): %s" % precision_with_high_threshold
print "Recall (threshold = 0.9)   : %s" % recall_with_high_threshold
```

```
Precision (threshold = 0.9): 0.969527896996
Recall (threshold = 0.9)   : 0.884463427656
```

Quiz Question (variant 1): Does the **precision** increase with a higher threshold?

Quiz Question (variant 2): Does the **recall** increase with a higher threshold?

Precision-recall curve

Now, we will explore various different values of tresholds, compute the precision and recall scores, and then plot the precision-recall curve.

In [30]:

```
threshold_values = np.linspace(0.5, 1, num=100)
print threshold_values
```

```
[ 0.5         0.50505051  0.51010101  0.51515152  0.52020202  0.52525253
  0.53030303  0.53535354  0.54040404  0.54545455  0.55050505  0.55555556
  0.56060606  0.56565657  0.57070707  0.57575758  0.58080808  0.58585859
  0.59090909  0.5959596   0.6010101   0.60606061  0.61111111  0.61616162
  0.62121212  0.62626263  0.63131313  0.63636364  0.64141414  0.64646465
  0.65151515  0.65656566  0.66161616  0.66666667  0.67171717  0.67676768
  0.68181818  0.68686869  0.69191919  0.6969697   0.7020202   0.70707071
  0.71212121  0.71717172  0.72222222  0.72727273  0.73232323  0.73737374
  0.74242424  0.74747475  0.75252525  0.75757576  0.76262626  0.76767677
  0.77272727  0.77777778  0.78282828  0.78787879  0.79292929  0.7979798
  0.8030303   0.80808081  0.81313131  0.81818182  0.82323232  0.82828283
  0.83333333  0.83838384  0.84343434  0.84848485  0.85353535  0.85858586
  0.86363636  0.86868687  0.87373737  0.87878788  0.88383838  0.88888889
  0.89393939  0.8989899   0.9040404   0.90909091  0.91414141  0.91919192
  0.92424242  0.92929293  0.93434343  0.93939394  0.94444444  0.94949495
  0.95454545  0.95959596  0.96464646  0.96969697  0.97474747  0.97979798
  0.98484848  0.98989899  0.99494949  1.        ]
```

For each of the values of threshold, we compute the precision and recall scores.

In [31]:

```
precision_all = []
recall_all = []

probabilities = model.predict(test_data, output_type='probability')
for threshold in threshold_values:
    predictions = apply_threshold(probabilities, threshold)

    precision = graphlab.evaluation.precision(test_data['sentiment'], predictions)
    recall = graphlab.evaluation.recall(test_data['sentiment'], predictions)

    precision_all.append(precision)
    recall_all.append(recall)
```

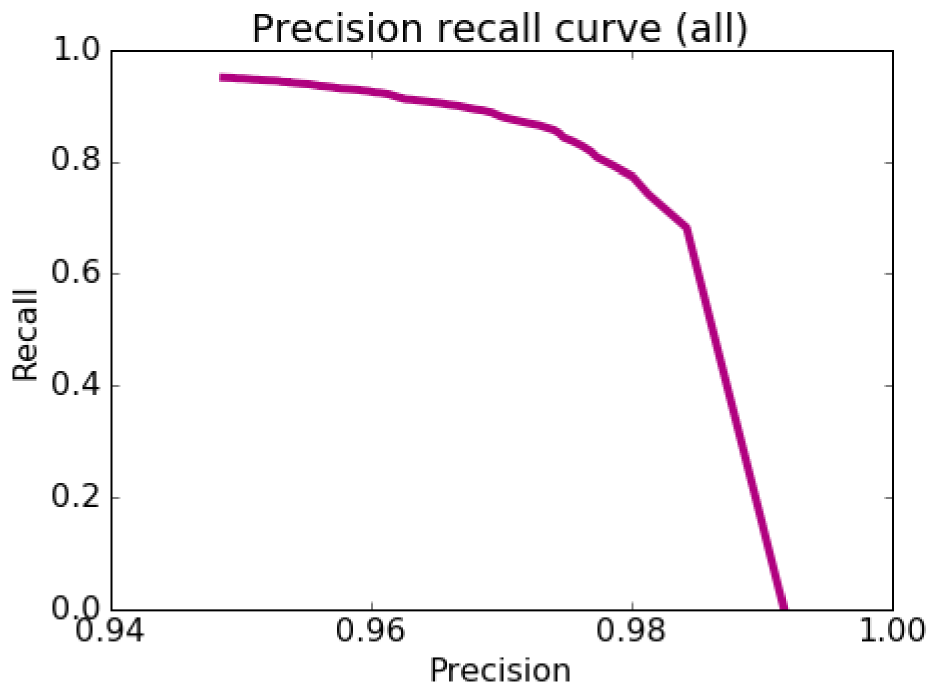
Now, let's plot the precision-recall curve to visualize the precision-recall tradeoff as we vary the threshold.

In [32]:

```
import matplotlib.pyplot as plt
%matplotlib inline

def plot_pr_curve(precision, recall, title):
    plt.rcParams['figure.figsize'] = 7, 5
    plt.locator_params(axis = 'x', nbins = 5)
    plt.plot(precision, recall, 'b-', linewidth=4.0, color = '#B0017F')
    plt.title(title)
    plt.xlabel('Precision')
    plt.ylabel('Recall')
    plt.rcParams.update({'font.size': 16})

plot_pr_curve(precision_all, recall_all, 'Precision recall curve (all)')
```



Quiz Question: Among all the threshold values tried, what is the **smallest** threshold value that achieves a precision of 96.5% or better? Round your answer to 3 decimal places.

In [33]:

```
threshold_values[np.array(precision_all) >= 0.965].min()
```

Out[33]:

```
0.838383838383845
```

Quiz Question: Using `threshold` = 0.98, how many **false negatives** do we get on the **test_data**? (**Hint**: You may use the `graphlab.evaluation.confusion_matrix` function implemented in GraphLab Create.)

In [34]:

```
threshold = 0.98
predictions = apply_threshold(probabilities, threshold)
confusion_matrix = graphlab.evaluation.confusion_matrix(test_data['sentiment'], predictions
confusion_matrix
```

Out[34]:

target_label	predicted_label	count
1	-1	5826
1	1	22269
-1	-1	4754
-1	1	487

[4 rows x 3 columns]

In [35]:

```
confusion_matrix[(confusion_matrix['target_label'] == +1) & (confusion_matrix['predicted_la
```

Out[35]:

```
5826L
```

This is the number of false negatives (i.e the number of reviews to look at when not needed) that we have to deal with using this classifier.

Evaluating specific search terms

So far, we looked at the number of false positives for the **entire test set**. In this section, let's select reviews using a specific search term and optimize the precision on these reviews only. After all, a manufacturer would be interested in tuning the false positive rate just for their products (the reviews they want to read) rather than that of the entire set of products on Amazon.

Precision-Recall on all baby related items

From the **test set**, select all the reviews for all products with the word 'baby' in them.

In [36]:

```
baby_reviews =  test_data[test_data['name'].apply(lambda x: 'baby' in x.lower())]
```

Now, let's predict the probability of classifying these reviews as positive:

In [37]:

```
probabilities = model.predict(baby_reviews, output_type='probability')
```

Let's plot the precision-recall curve for the **baby_reviews** dataset.

First, let's consider the following `threshold_values` ranging from 0.5 to 1:

In [38]:

```
threshold_values = np.linspace(0.5, 1, num=100)
```

Second, as we did above, let's compute precision and recall for each value in `threshold_values` on the **baby_reviews** dataset. Complete the code block below.

In [39]:

```
precision_all = []
recall_all = []

for threshold in threshold_values:

    # Make predictions. Use the `apply_threshold` function
    ## YOUR CODE HERE
    predictions = apply_threshold(probabilities, threshold)

    # Calculate the precision.
    # YOUR CODE HERE
    precision = graphlab.evaluation.precision(baby_reviews['sentiment'], predictions)

    # YOUR CODE HERE
    recall = graphlab.evaluation.recall(baby_reviews['sentiment'], predictions)

    # Append the precision and recall scores.
    precision_all.append(precision)
    recall_all.append(recall)
```

Quiz Question: Among all the threshold values tried, what is the **smallest** threshold value that achieves a precision of 96.5% or better for the reviews of data in **baby_reviews**? Round your answer to 3 decimal places.

In [40]:

```
threshold_values[np.array(precision_all) >= 0.965].min()
```

Out[40]:

```
0.86363636363636365
```

Quiz Question: Is this threshold value smaller or larger than the threshold used for the entire dataset to achieve the same specified precision of 96.5%?

smallest threshold yielding precision > 0.965 (calc'd from entire dataset) : 0.838

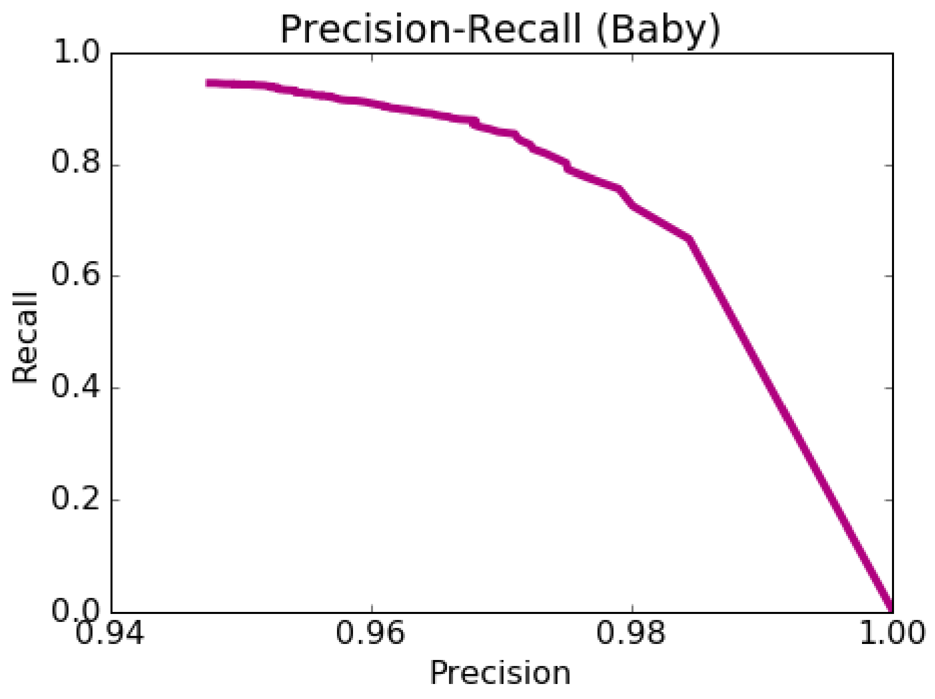
smallest threshold yielding precision > 96.5% (calc'd from baby_reviews dataset) : 0.864

answer = LARGER!!!

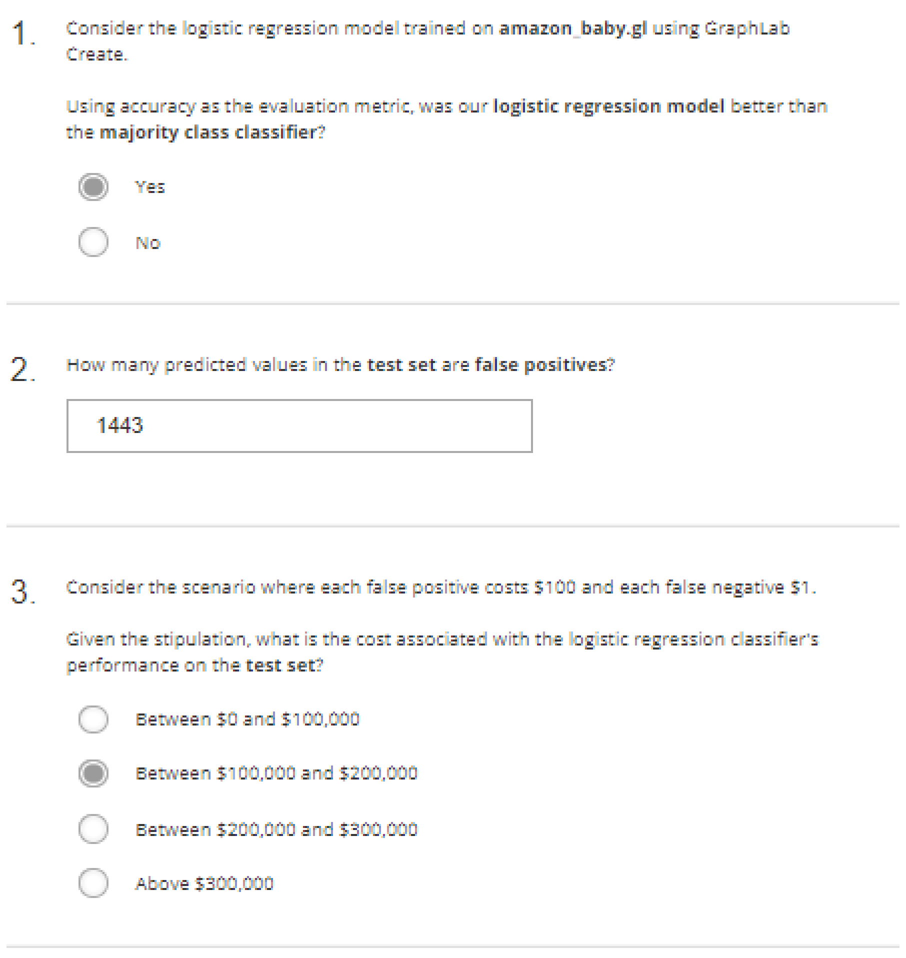
Finally, let's plot the precision recall curve.

In [41]:

```
plot_pr_curve(precision_all, recall_all, "Precision-Recall (Baby)")
```



Quiz



1. Consider the logistic regression model trained on **amazon_baby.gl** using GraphLab Create.

   Using accuracy as the evaluation metric, was our **logistic regression model** better than the **majority class classifier**?

   [✓ Yes]

   No

2. How many predicted values in the **test set** are **false positives**?

   1443

3. Consider the scenario where each false positive costs $100 and each false negative $1.

   Given the stipulation, what is the cost associated with the logistic regression classifier's performance on the **test set**?

   Between $0 and $100,000

   [✓ Between $100,000 and $200,000]

   Between $200,000 and $300,000

   Above $300,000

4. Out of all reviews in the **test set** that are predicted to be positive, what fraction of them are **false positives**? (Round to the second decimal place e.g. 0.25)

0.04

5. Based on what we learned in lecture, if we wanted to reduce this fraction of false positives to be below 3.5%, we would:

- Discard a sufficient number of positive predictions
- Discard a sufficient number of negative predictions
- [✓ Increase threshold for predicting the positive class ($\hat{y} = +1$)]
- Decrease threshold for predicting the positive class ($\hat{y} = +1$)

6. What fraction of the positive reviews in the **test_set** were correctly predicted as positive by the classifier? Round your answer to 2 decimal places.

0.95

7. What is the recall value for a classifier that predicts **+1** for all data points in the **test_data**?

1

8. What happens to the number of positive predicted reviews as the threshold increased from 0.5 to 0.9?

- More reviews are predicted to be positive.
- [✓ Fewer reviews are predicted to be positive.]

9. Consider the metrics obtained from setting the threshold to 0.5 and to 0.9.

Does the **precision** increase with a higher threshold?

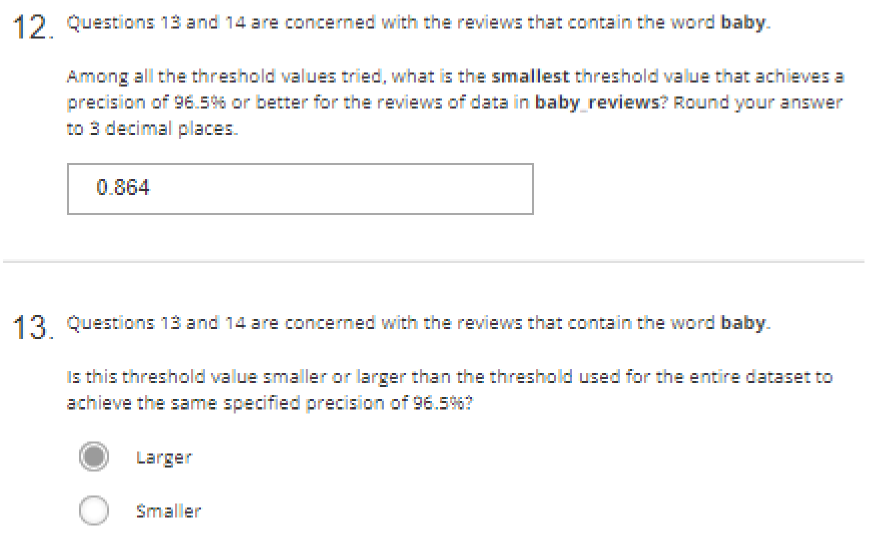
- [✓ Yes]
- No

10. Among all the threshold values tried, what is the **smallest** threshold value that achieves a precision of 96.5% or better? Round your answer to 3 decimal places.

0.838

11. Using threshold = 0.98, how many **false negatives** do we get on the **test_data**? (**Hint**: You may use the graphlab.evaluation.confusion_matrix function implemented in GraphLab Create.)

5826



12. Questions 13 and 14 are concerned with the reviews that contain the word **baby**.

Among all the threshold values tried, what is the **smallest** threshold value that achieves a precision of 96.5% or better for the reviews of data in **baby_reviews**? Round your answer to 3 decimal places.

0.864

13. Questions 13 and 14 are concerned with the reviews that contain the word **baby**.

Is this threshold value smaller or larger than the threshold used for the entire dataset to achieve the same specified precision of 96.5%?

[✓ Larger]

Smaller