Exploring precision and recall

The goal of this assignment 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.

If you are doing the assignment with IPython Notebook

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

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

What you need to download

If you are using GraphLab Create:

- Download the Amazon product review dataset in SFrame format: amazon_baby.gl.zip
- Download the companion IPython notebook: module-9-precision-recall-assignment-blank.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

If you are not using GraphLab Create:

- If you are using SFrame, download the Amazon product review dataset in SFrame format: amazon_baby.gl.zip
- If you are using a different package, download the Amazon product review dataset in CSV format: amazon_baby.csv.zip

If you are using GraphLab Create and the companion IPython Notebook

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

If you are using other tools

This section is designed for people using tools other than GraphLab Create. Even though some instructions are specific to scikit-learn, most part of the assignment should be applicable to other tools as well. However, **we highly suggest you use SFrame since it is open source.** In this part of the assignment, we describe general instructions, however we will tailor the instructions for SFrame and scikit-learn.

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

Load Amazon dataset

1. Load the dataset consisting of baby product reviews on Amazon.com. Store the data in a data frame **products**. In SFrame, you would run

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

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

```
pip install sframe
```

Perform text cleaning

2. We start by removing punctuation, so that words "cake." and "cake!" are counted as the same word.

- Write a function **remove_punctuation** that strips punctuation from a line of text
- Apply this function to every element in the **review** column of **products**, and save the result to a new column **review_clean**.

Refer to your tool's manual for string processing capabilities. Python lets us express the operation in a succinct way, as follows:

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

products['review_clean'] = products['review'].apply(remove_punctuation)
```

Aside. In this notebook, we remove all punctuation for the sake of simplicity. A smarter approach to punctuation would preserve phrases such as "I'd", "would've", "hadn't" and so forth. See this page for an example of smart handling of punctuation.

Extract Sentiments

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

```
products = products[products['rating'] != 3]
```

4. 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. A good way is to create an anonymous function that converts a rating into a class label and then apply that function to every element in the **rating** column. In SFrame, you would use apply():

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

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

Split into training and test sets

5. 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. If you are using SFrame, make sure to use seed=1 so that you get the same result as everyone else does. (This way, you will get the right numbers for the quiz.) If you are other tools, consult appropriate manuals to perform a train/test split.

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

If you are not using SFrame, download the list of indices for the training and test sets: module-9-assignment-train-idx.json, module-9-assignment-test-idx.json. IMPORTANT: If you are using a programming language with 1-based indexing (e.g. R, Matlab), make sure to increment all indices by 1.

Call the training and test sets **train_data** and **test_data**, respectively.

Build the word count vector for each review

6. We will now compute the word count for each word that appears in the reviews. A vector consisting of word counts is often referred to as **bag-of-word features**. Since most words occur in only a few reviews, word count vectors are sparse. For this reason, scikit-learn and many other tools use sparse matrices to store a collection of word count vectors. Refer to appropriate manuals to produce sparse word count vectors. General steps for extracting word count vectors are as follows:

- Learn a vocabulary (set of all words) from the training data. Only the words that show up in the training data will be considered for feature extraction.
- Compute the occurrences of the words in each review and collect them into a row vector.
- Build a sparse matrix where each row is the word count vector for the corresponding review. Call this matrix **train_matrix**.
- Using the same mapping between words and columns, convert the test data into a sparse matrix **test_matrix**.

The following cell uses CountVectorizer in scikit-learn. Notice the **token_pattern** argument in the constructor.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(token_pattern=r'\b\w+\b')
     # Use this token pattern to keep single-letter words
# First, learn vocabulary from the training data and assign columns to words
# Then convert the training data into a sparse matrix
train_matrix = vectorizer.fit_transform(train_data['review_clean'])
# Second, convert the test data into a sparse matrix, using the same word-column mapping
test_matrix = vectorizer.transform(test_data['review_clean'])
```

Keep in mind that the test data must be transformed in the same way as the training data.

Train a sentiment classifier with logistic regression

7. Learn a logistic regression classifier using the training data. If you are using scikit-learn, you should create an instance of the LogisticRegression class and then call the method fit() to train the classifier. This model should use the sparse word count matrix (**train_matrix**) as features and the column **sentiment** of **train_data** as the target. Use the default values for other parameters. Call this model **model**.

Model Evaluation

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

Accuracy

8. 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

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

Compute the accuracy on the test set using your tool of choice. If you are using scikit-learn, you can use the pre-defined method accuracy_score:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true=test_data['sentiment'].to_numpy(), y_pred=model.predict(test
_matrix))
print "Test Accuracy: %s" % accuracy
```

Baseline: Majority class prediction

9. 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 is simply the fraction of positive reviews in the test set:

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

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

Confusion Matrix

10. 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  |
+-------+-----+----------------------+----------------------+
```

Using your tool, print out the confusion matrix for a classifier. For instance, scikit-learn provides the method confusion_matrix for this purpose:

```
from sklearn.metrics import confusion_matrix
cmat = confusion_matrix(y_true=test_data['sentiment'].to_numpy(),
                        y_pred=model.predict(test_matrix),
                        labels=model.classes_)    # use the same order of class as the LR mod
el.
print ' target_label | predicted_label | count '
print '--------------+-----------------+-------'
# Print out the confusion matrix.
# NOTE: Your tool may arrange entries in a different order. Consult appropriate manuals.
for i, target_label in enumerate(model.classes_):
    for j, predicted_label in enumerate(model.classes_):
        print '{0:^13} | {1:^15} | {2:5d}'.format(target_label, predicted_label, cmat[i,j])
```

IMPORTANT. In one way or another, make sure to print out the predicted label and the true label for each and every entry of the confusion matrix. This way, we don't mistake one type of mistake for another. The cell above produces the following output:

```
 target_label | predicted_label | count 
--------------+-----------------+-------
     -1       |       -1        |  3787 
     -1       |        1        |  1454 
      1       |       -1        |   805 
      1       |        1        | 27290 
```

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

Computing the cost of mistakes

11. 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:

- $100 for each false positive.
- $1 for each false negative.
- 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**?

Precision and Recall

12. 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:

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

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**. Scikit-learn provides a predefined method for computing precision. (Consult appropriate manuals if you are using other tools.)

```
from sklearn.metrics import precision_score
precision = precision_score(y_true=test_data['sentiment'].to_numpy(),
                            y_pred=model.predict(test_matrix))
print "Precision on test data: %s" % precision
```

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)

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 quiz)

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

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

Let us compute the recall on the **test_data**. Scikit-learn provides a predefined method for computing recall as well. (Consult appropriate manuals if you are using other tools.)

```
from sklearn.metrics import recall_score
recall = recall_score(y_true=test_data['sentiment'].to_numpy(),
                      y_pred=model.predict(test_matrix))
print "Recall on test data: %s" % recall
```

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

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

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

14. 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 array, where each element is set to +1 or -1 depending whether the corresponding probability exceeds threshold.

15. Using the **model** you trained, compute the class probability values P(y=+1|x,w) for the data points in the **test_data**. Then use thresholds set at 0.5 (default) and 0.9 to make predictions from these probability values.

Note. If you are using scikit-learn, make sure to use **predict_proba**() function, not decision_function(). Also, note that the predict_proba() function returns the probability values for both classes +1 and -1. So make sure to extract the second column, which correspond to the class +1.

```
probabilities = model.predict_proba(test_matrix)[:,1]
```

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

Exploring the associated precision and recall as the threshold varies

16. By changing the probability threshold, it is possible to influence precision and recall. Compute precision and recall for threshold values 0.5 and 0.9.

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

17. Now, we will explore various different values of tresholds, compute the precision and recall scores, and then plot the precision-recall curve. Use 100 equally spaced values between 0.5 and 1. In Python, we run

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

For each of the values of threshold, we first obtain class predictions using that threshold and then compute the precision and recall scores. Save the precision scores and recall scores to lists **precision_all** and **recall_all**, respectively.

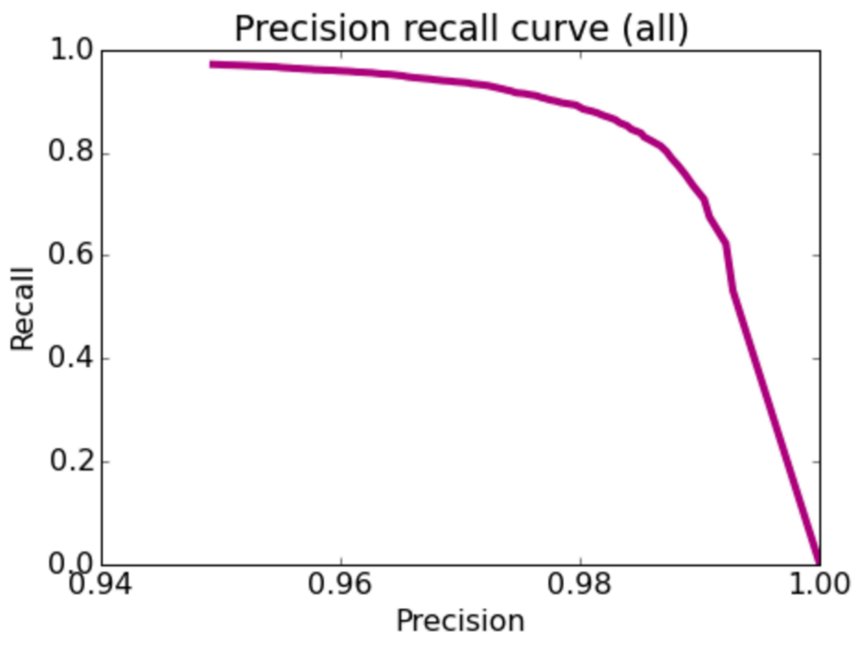
18. Let's plot the precision-recall curve to visualize the precision-recall tradeoff as we vary the threshold. Implement the function **plot_pr_curve** that generates a connected scatter plot from the lists of precision and recall scores. The function would be implemented in matplotlib as follows; for other tools, consult appropriate manuals.

```
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})
```

19. Once the function **plot_pr_curve** is complete, plot the precision-recall curve for the test set by running

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

You should obtain a connected scatter plot that looks like this figure:



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.

Quiz Question: Using threshold = 0.98, how many **false negatives** do we get on the **test_data**? 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

20. From the test set, select all the reviews for all products with the word 'baby' in them. If you are using SFrame, generate a binary mask with apply() and index **test_data** with the mask. Save the selection to a new data frame named **baby_reviews**.

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

21. Now, let's predict the probability of classifying these reviews as positive. Make sure to convert the column **review_clean** of **baby_reviews** into a 2D array before computing class probability values. In scikit-learn, this task would be implemented as follows:

```
baby_matrix = vectorizer.transform(baby_reviews['review_clean'])
probabilities = model.predict_proba(baby_matrix)[:,1]
```

22. Let's plot the precision-recall curve for the **baby_reviews** dataset. We again use 100 equally spaced values between 0.5 and 1 for the threshold.

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

For each of the values of threshold, we first obtain class predictions for **baby_reviews** using that threshold. Then we compute the precision and recall scores for **baby_reviews**. Save the precision scores and recall scores to lists **precision_all** and **recall_all**, respectively.

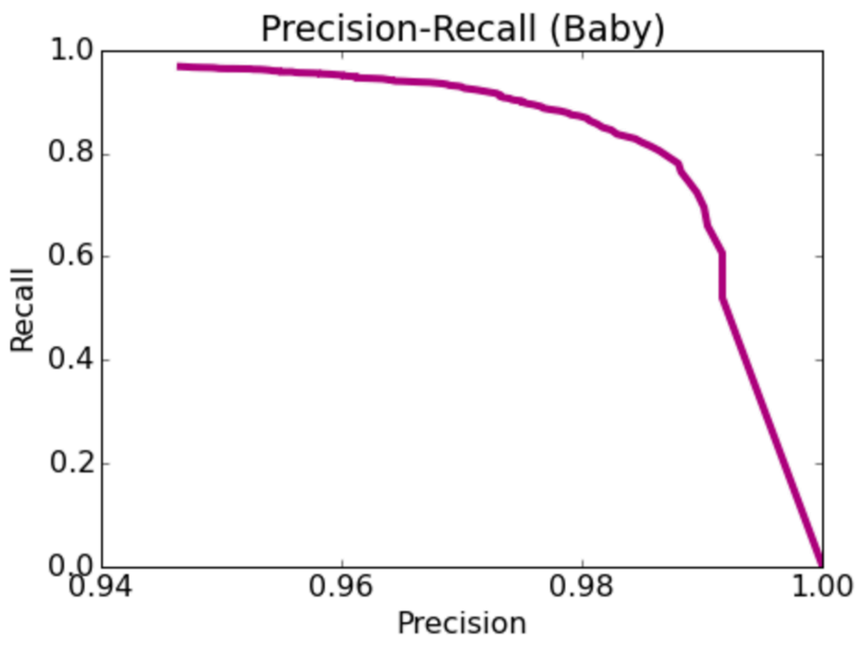
Quiz Question: Among all the threshold values tried, what is the **smallest** threshold value that achieves a precision of 96.5% or better for the reviews of data in **baby_reviews**? Round your answer to 3 decimal places.

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%?

23. Plot the precision-recall curve for **baby_reviews** only by running

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

You should obtain a figure similar to the following:


Precision-Recall (Baby)
Recall
Precision