Predicting sentiment from product reviews

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

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

- 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.

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.

What you need to download

If you are using GraphLab Create:

- Download the Amazon product review data in SFrame format: amazon_baby.gl.zip
- Download the companion IPython Notebook: module-2-linear-classifier-assignment-blank.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.
- Follow the instructions contained in the IPython notebook.

If you are not using GraphLab Create

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

IMPORTANT. Make sure to fill n/a values in the **review** column with empty strings (if applicable). The n/a values indicate empty reviews. For instance, Pandas's fillna() method lets you replace all N/As in the **review** columns as follows:

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

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.)

```
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-2-assignment-train-idx.json, module-2-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'\S+')
# 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

We will now use logistic regression to create a sentiment classifier on the training data.

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 **sentiment_model**.

8. There should be over 100,000 coefficients in this **sentiment_model**. Recall from the lecture that positive weights w_j correspond to weights that cause positive sentiment, while negative weights correspond to negative sentiment. Calculate the number of positive (>= 0), which is actually nonnegative) coefficients.

Quiz question: How many weights are >= 0?

Making predictions with logistic regression

9. 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 data points in the test data. Take the 11th, 12th, and 13th data points in the test data and save them to **sample_test_data**. The following cell extracts the three data points from the SFrame **test_data** and print their content:

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

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

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

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 rating (-1), the review is quite negative.

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

10. 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:

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

where $h(x_i)$ represents the features for data point i. We will write some code to obtain the scores. For each row, the score (or margin) is a number in the range $(-\inf, \inf)$. Use a pre-built function in your tool to calculate the score of each data point in **sample_test_data**. In scikit-learn, you can call the decision_function() function.

Hint: You'd probably need to convert sample_test_data into the sparse matrix format first.

```
sample_test_matrix = vectorizer.transform(sample_test_data['review_clean'])
scores = sentiment_model.decision_function(sample_test_matrix)
print scores
```

Predicting Sentiment

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

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

Using scores, write code to calculate predicted labels for **sample_test_data**.

Checkpoint: Make sure your class predictions match with the ones obtained from **sentiment_model**. The logistic regression classifier in scikit-learn comes with the **predict** function for this purpose.

Probability Predictions

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

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

Using the 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]**.

Checkpoint: Make sure your probability predictions match the ones obtained from **sentiment_model**.

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

13. We now turn to examining the full test dataset, **test_data**, and use sklearn.linear_model.LogisticRegression to form predictions on all of the test data points.

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**.
2. Sort the data according to those predictions and pick the top 20.

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

14. 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**.

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

Compute accuracy of the classifier

15. 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 **sentiment_model** to compute class predictions.
- **Step 2:** Count the number of data points when the predicted class labels match the ground truth labels.
- **Step 3:** Divide the total number of correct predictions by the total number of data points in the dataset.

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

16. 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 20 words to work with. These are:

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

Compute a new set of word count vectors using only these words. The CountVectorizer class has a parameter that lets you limit the choice of words when building word vectors:

```
vectorizer_word_subset = CountVectorizer(vocabulary=significant_words) # limit to 20 words
train_matrix_word_subset = vectorizer_word_subset.fit_transform(train_data['review_clean'])
test_matrix_word_subset = vectorizer_word_subset.transform(test_data['review_clean'])
```

Compute word count vectors for the training and test data and obtain the sparse matrices **train_matrix_word_subset** and **test_matrix_word_subset**, respectively.

Train a logistic regression model on a subset of data

17. Now build a logistic regression classifier with **train_matrix_word_subset** as features and **sentiment** as the target. Call this model **simple_model**.

18. Let us inspect the weights (coefficients) of the **simple_model**. First, build a table to store (word, coefficient) pairs. If you are using SFrame with scikit-learn, you can combine words with coefficients by running

```
simple_model_coef_table = sframe.SFrame({'word':significant_words,
                                         'coefficient':simple_model.coef_.flatten()})
```

Sort the data frame by the coefficient value in descending order.

Note: Make sure that the intercept term is excluded from this table.

Quiz Question. Consider the coefficients of **simple_model**. How many of the 20 coefficients (corresponding to the 20 **significant_words**) are positive for the **simple_model**?

Quiz Question. Are the positive words in the **simple_model** also positive words in the **sentiment_model**?

Comparing models

19. We will now compare the accuracy of the **sentiment_model** and the **simple_model**.

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

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

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

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

Next, compute the classification accuracy of the **simple_model** on the **test_data**.

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

Baseline: Majority class prediction

21. 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.

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).

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