**King Fahd University of Petroleum & Minerals**

College of Computer Sciences and Engineering

Information and Computer Science Department

**ICS 490: Special Topics in Computer Science**

Programming Assignment No. 3

**Classification of Amazon Product Reviews Using Logistic Regression**

**Posted:** Tuesday 28th November 2017  
**Deadline:** Monday 04th December 2017 @ 11:59 PM (Before Midnight)

The goal of this assignment is to classify Amazon product reviews using the logistic regression classifier. You will:

Extract features from Amazon product reviews.

1. Convert an SFrame into a NumPy array.
2. Implement the link function for logistic regression.
3. Write a function to compute the derivative of the log likelihood function with respect to a single coefficient.
4. Implement gradient ascent.
5. Given a set of coefficients, predict sentiments.
6. Compute classification accuracy for the logistic regression model.

**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 the latest version of GraphLab Create.

1. **What you need to download**

**Note:** You will need to download the zipped file **prog\_assgt\_4\_data\_sets.rar** which contains all necessary data files for completing this programming assignment.

**A.1 Data Preparation:**

1. If you are using a different package, extract the Amazon product review dataset (subset) in CSV format: **amazon\_baby\_subset.csv.zip**
2. Extract the list of **193 significant words**: **important\_words.json.zip**
3. Load review dataset
4. For this assignment, we will use a subset of the Amazon product review dataset. The subset was chosen to contain similar numbers of positive and negative reviews, as the original dataset consisted primarily of positive reviews.
5. Load the dataset into a data frame named **products**. One column of this dataset is **sentiment**, corresponding to the class label with **+1** indicating a review with positive sentiment and **-1** for negative sentiment.
6. Let us quickly explore more of this dataset. The **name** column indicates the name of the product. Try listing the name of the first 10 products in the dataset.
7. After that, try counting the number of positive and negative reviews.

**Note:** For this assignment, we eliminated **class imbalance** by choosing a subset of the data with a similar number of positive and negative reviews.

1. Apply text cleaning on the review data. We will perform some simple feature cleaning using data frames.
2. We compiled a list of 193 most frequent words into the JSON file named **important\_words.json**. Load the words into a list **important\_words**.
3. Let us perform 2 simple data transformations:
   1. Remove punctuation: If your tool supports it, fill n/a values in the **review** column with empty strings. The n/a values indicate empty reviews. For instance, **Pandas**'s **fillna()** method lets you replace all N/A's in the **review** columns as follows:

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

Write a function **remove\_punctuation** that takes a line of text and removes all punctuation from that text. The function should be analogous to the following Python code:

**def remove\_punctuation(text):**

**import string**

**return text.translate(None, string.punctuation)**

Apply the **remove\_punctuation** function on every element of the **review** column and assign the result to the new column **review\_clean**.

**Note**: Many data frame packages support **apply** operation for this type of task. Consult appropriate manuals.

* 1. Compute word counts (only for important\_words): For each word in **important\_words**, we compute a count for the number of times the word occurs in the review. We will store this count in a separate column (one for each word). The result of this feature processing is a single column for each word in **important\_words** which keeps a count of the number of times the respective word occurs in the review text.

**Note:** There are several ways of doing this. One way is to create an anonymous function that counts the occurrence of a particular word and apply it to every element in the **review\_clean** column. Repeat this step for every word in **important\_words**. Your code should be analogous to the following:

**for word in important\_words:**

**products[word] = products['review\_clean'].apply(lambda s : s.split().count(word))**

Now, the data frame **products** should contain one column for each of the 193 **important\_words**. As an example, the column **perfect** contains a count of the number of times the word **perfect** occurs in each of the reviews.

Now, write some code to compute the number of product reviews that contain the word perfect.

**Hint:**

* First create a column called **contains\_perfect** which is set to 1 if the count of the word **perfect** (stored in column perfect is >= 1.
* Sum the number of 1s in the column **contains\_perfect**.

**Analysis Question 1:** How many reviews contain the word perfect?

1. Convert data frame to multi-dimensional array: NumPy is a good choice. Write a function that extracts columns from a data frame and converts them into a multi-dimensional array. We plan to use them throughout the course, so make sure to get this function right.

The function should accept three parameters:

* **dataframe**: a data frame to be converted
* **features**: a list of string, containing the names of the columns that are used as features.
* **label**: a string, containing the name of the single column that is used as class labels.

The function should return two values:

* one 2D array for features
* one 1D array for class labels

The function should do the following:

* Prepend a new column **constant** to **dataframe** and fill it with 1's. This column takes account of the intercept term. Make sure that the constant column appears first in the data frame.
* Prepend a string 'constant' to the list **features**. Make sure the string 'constant' appears first in the list.
* Extract columns in **dataframe** whose names appear in the list **features**.
* Convert the extracted columns into a 2D array using a function in the data frame library. If you are using Pandas, you would use **as\_matrix()** function.
* Extract the single column in **dataframe** whose name corresponds to the string **label**.
* Convert the column into a 1D array.
* Return the 2D array and the 1D array.

If using Pandas, you would execute these steps as follows:

**def get\_numpy\_data(dataframe, features, label):**

**dataframe['constant'] = 1**

**features = ['constant'] + features**

**features\_frame = dataframe[features]**

**feature\_matrix = features\_frame.as\_matrix()**

**label\_sarray = dataframe[label]**

**label\_array = label\_sarray.as\_matrix()**

**return(feature\_matrix, label\_array)**

1. Using the function written above, extract two arrays **feature\_matrix** and **sentiment**. The 2D array **feature\_matrix** would contain the content of the columns given by the list **important\_words**. The 1D array **sentiment** would contain the content of the column **sentiment**.

**Analysis Question 2:** How many features are there in the **feature\_matrix**?

**Analysis Question 3:** Assuming that the intercept is present, how does the number of features in **feature\_matrix** relate to the number of features in the logistic regression model?

1. Use the logistic classifier of scikit-learn package to predict the sentiment of the Amazon product reviews.

**Analysis question 5:** How many reviews were predicted to have positive sentiment?

**Classification Accuracy:** The classification accuracy of any classifier is defined as follows:



**Analysis question 6:** What is the accuracy of the model on predictions made above? (round to 2 digits of accuracy)

1. We will define the "**most positive words**". These are words that correspond most strongly with positive reviews. In order to do this, we will first do the following:

* Treat each coefficient as a tuple, i.e. (**word, coefficient\_value**). The intercept has no corresponding word, so throw it out.
* Sort all the (**word**, **coefficient\_value**) tuples by **coefficient\_value** in descending order. Save the sorted list of tuples to **word\_coefficient\_tuples**.

Your code should be analogous to the following:

**coefficients = list(coefficients[1:]) # exclude intercept**

**word\_coefficient\_tuples = [(word, coefficient) for word, coefficient in zip(important\_words, coefficients)]**

**word\_coefficient\_tuples = sorted(word\_coefficient\_tuples, key=lambda x:x[1], reverse=True)**

Now, **word\_coefficient\_tuples** contains a sorted list of (**word**, **coefficient\_value**) tuples. The first 10 elements in this list correspond to the words that are most positive.

15. Compute the 10 words that have the most positive coefficient values. These words are associated with positive sentiment.

**Analysis question 7:** Which word is not present in the top 10 "most positive" words?

16. Next, we repeat this exercise on the 10 most negative words. That is, we compute the 10 words that have the most negative coefficient values. These words are associated with negative sentiment.

**Analysis question 8:** Which word is **not** present in the top 10 "most negative" words?