ASSIGNMENT 2

Inputs

The input to this assignment was the product reviews taken from Amazon.com from many product types. The dataset can be found here [http://www.cs.jhu.edu/~mdredze/datasets/sentiment/](http://www.cs.jhu.edu/~mdredze/datasets/sentiment/). The entire data set was divided into 4 ratings (1,2,4,5). The tokenized version of the dataset was available on [http://www.cs.berkeley.edu/~jfc/DataMining/SP12/restricted/HW2/books/](http://www.cs.berkeley.edu/~jfc/DataMining/SP12/restricted/HW2/books/).

Approach

Training Set:

We are constructing the training data set as a matrix ($M$) of **document id** (rows) x **unique tokens** (columns). Each $i,j$ value from the matrix represents the count of the token from the particular document. To normalize the token counts we have used a weight matrix which was constructed as **unique tokens** (rows) x 1. Each term weight for the token was calculated as follows:

$$
\text{Token weight (Tw)} = \frac{\text{total count of the token across corpus}}{\text{total count of the words across corpus}}
$$

So the final formula we used can be represented as follows:

$$
Y = X \times \beta + \epsilon
$$

Where $X = X(\text{count matrix}) \times Tw$.

Thus $X$ indicates the weight of the documents with respect to the entire corpus.

For this model, $Y$ is our rating matrix represented as **documents** (rows) x 1. The value for $\beta$ is represented as $[\beta_0 \beta_1]^T$. To achieve these values for $\beta$, we prefix the final input matrix $X$ with unit column vector of size documents x 1 ($X = [1 \ X(\text{count matrix})]$).

Test Set:

Test Set comprises of 10% of the documents collected at random using the `randperm` function of Matlab. E.g. `randRows=randperm(nRows);` The other 90% of the documents are contained within the training set.

Stemming:

For stemming we iterate through all the unique tokens and use `porterStemmer` to get the stemmed form of the word and attach that word onto a new column. In addition to this we have a vector which has counts of all the tokens across the training corpus. We now iterate through this vector of tokens and find the common tokens and take a summation of those counts and replace that value where we found common tokens. So, that we can ultimately calculate weights of those tokens based on their aggregated count. Now we can multiply our new weight values with...
our original matrix **document id**(rows) x **unique tokens** and get improved X values.

**Stopwords:**

We have imported the list of stopwords from [http://www.tomdiethe.com/teaching/englishST.txt](http://www.tomdiethe.com/teaching/englishST.txt). These stopwords are then compared with the tokens from the **dict.smap** provided and the corresponding row (index being the token code) is assigned a term weight 0 in the term weight matrix **Tw**. This ensures that the contribution of the stop weight is eliminated from the final score of the document. In addition to the above stop words we have also added following tags to the stop words list:

**Stochastic Gradient:**

Once we get the initial values of $\beta$, we refine it further to obtain more precise values to make the linear regression curve as close as to the actual curve as possible.

for the purpose of this assignment , we saw that since the number of review documents was heavily skewed towards ratings 4,5 – we took all review documents of rating 1 and 2 and treated 90% of them as a training set to calculate our beta and the remaining as our test set.

**Implementation:**

**Attempt 1: Scala**

We tried implementing the above explained approaches in Scala. We encountered the little endian problem while reading the binary file. We overcame this problem with the following approach

```scala
while ( shiftBy < 32){
    accum |= ( dis.readByte() & 0xff ) << shiftBy;
    shiftBy += 8
}
```

We successfully loaded the file and created 3 hashmaps (for `token_word` -> `token_id`, `token_id` -> `token_word`, `token_id`->`count`). But we got several errors while creating a sparse vector with the given inputs because of which we switched to Matlab.

**Attempt 2: Matlab**

Initially we tried creating the $X$ by creating a vector for each document and then concatenating it vertically with the main vector. We then switched to creating a sparse vector filled with zeros (using spalloc method) and then filling in count
values for each of the token in the document by using hist() function. Both these methods were time consuming.

Finally, we created the matrix using spconvert function of matlab which helped us compute the matrix for the entire corpus in about an hour.

**Analysis**

We observed the dataset to be quite biased with more reviews for rating 4 and 5. They accounted for more than 70% of the dataset. This would skew the entire predictions towards rating 4.

```matlab
rating1Indexes = find(Y==1);
resultX1 = X(rating1Indexes,:)
```

The data is as follows:

Hence we decided to modify our term weight matrix which was initially defined as

Token weight ($T_w$) = total count of the token across corpus / total count of the words across corpus

To

Token weight ($T_w$) = total count of the token across corpus / total count of the words across the reviews with the same rating

initially, the beta calculated using the Matrix M over all review documents was

$$\text{Beta} = \begin{bmatrix} 4.3508 & -0.0184 \end{bmatrix}$$

using this Beta to predict the ratings predicted a rating of 4 for a majority of the documents, which made us suspicious regarding the impact of reviews rated 4 and 5. On gathering a count of the documents by review, we found a 70% share of these reviews, confirming the bias in our rating predictions.

So we isolated our review documents with rating 1 and 2 and created the Matrix M comprising the tokens from those documents and their corresponding ratings.

So, using the $xTest12$ as our Matrix M over the training set, and ratingTest12 as the corresponding rating vector, we calculated the new Beta($\beta$)

$$\text{Beta}_{12} = \begin{bmatrix} 1.4035 & 0.0361 \end{bmatrix}$$

Applying the stochastic gradient method to refine our Beta over 20 iterations, we approached some convergence, using alpha= 0.000025
Beta-refined$_{12}$ = [1.4035 0.0361]

Stochastic gradient plot (for 5 iterations to refine Beta)

Note: ‘x’ marks the coordinates for each review(reviewId,rating).

Stemmer code Analysis:

We tried to incorporate the porter stemmer to try to stem all unique tokens from the dict.smap. However, we were encountering frequent OOM issues to pursue any experimentation, to capture new beta on this. Code is attached in stemmer.m