1 Overview

The major learning from this homework was dealing with performance hits and thinking about how to write efficient code for data to model computation. I found blocking highly helpful, and then was able to generate the unigram and bigram feature vectors within approximately 30-45 mins by using blocking, vectorization and efficient matrix computations. Another performance stumbling block was the hashing component to find the unique reviews. My first implementation of hashing was a very crude one and took a large amount of time. Later, however, I figured that using the hash bucket approach i.e. storing hashes in a hash table at the index \( \text{hash mod (some number)} \) is more efficient, since this considerably reduces the number of comparisons that have to be performed.

Models tried:

- Unigrams
- Exact Solution using Ridge Regression (minimize L2-norm loss function)
- L1 Loss function using Stochastic Gradient
- Working on Bigrams (not complete, but code attached)

Other notes:

- Due to the ordering of the reviews, I shuffled the feature matrix before input to the stochastic gradient computation.

**TODO:** Add a significance test to remove the coefficients that carry noise.

2 Constructing the Model:

To construct the model, I first read the complete tokens array from the .mat file into the memory. After that I constructed arrays to hold the markers for the tags: \(<\text{ratings}>\),
<review_text> and <review_text>. This was a fast operation because MATLAB can operate on the contiguous data in memory pretty fast. It’s also pretty clean code:

```matlab
mobj = matfile('tokenized.mat')
tokens = mobj.tokens(3,:);
Ratings = tokens(find(sparse(tokens==31))+1); % Vector of ratings
textBegin = find(sparse(tokens==41)); % Vector of <review_text> markers
textEnd = find(sparse(tokens==42));
```

Note that `textBegin(1)` will contain the start of the text of the first review, `textEnd(1)` the end of the text of the first review, and `Ratings(1)`, the rating of the first review.

Once this is done, I loop over the `textBegin` vector, each time picking up the corresponding `textEnd` value, and reading the block of memory from `textBegin+1` to `textEnd-1`. These are the tokens of this review. While looping over, I collect and build three vectors: `row`, `col`, `val`. The vector `row` contains the review numbers, `col` contains the token values, and `val` is just one for the tokens(i.e. `col` values) that occur in each review.

At the end of the looping, I build a sparse matrix using the MATLAB sparse command:

```matlab
X = sparse(row,col,val);
```

There is actually a layer on top of the above i.e. the blocking layer, since my machine didn’t have enough memory to store both the complete tokens array and the `row`, `col`, `val` vectors.

**Bigrams:** To construct the bigrams, I tried operating on the data in blocks of memory as well. Imagine a simple case where the features are (1 2 3) The possible bigrams are: (1—1, 1—2, 1—3, 2—1, 2—2, 2—3, 3—3). So one could imagine computing their indexes in a sparse matrix (filled with 1’s for at columns(indexes) of bigrams that do occur) in the following fashion:

```matlab
unigrams = tokens(textBegin(i)+1:textEnd(i)-1);
unigramsShifted = unigrams(2:length(unigrams));
unigrams = unigrams(1:length(unigrams)-1) - 1; % Trim the unigrams since last % element would not be used
% Then shift down the values % by 1, to get correct index % values
bigrams = unigrams*features + unigramsShifted;
```

### 3 Observations

#### 3.1 Exact Solution using Linear Regression

RMSE: 1.12
Features: Top 2000 highest frequency ones
Lambda (Ridge regularization constant): 2

I checked the beta weights vector by using the conditions mentioned in brackets below to get a sense of positive and negative indicators being used to predict the ratings:

*Words with highest beta weights* (>0.70)
Positive indicators: fine, strongly, humanity, minor, lovers, lake.
Bogus: hand, mary, husband, described, pain, central, structure, individuals

*Words with lowest beta weights* (<-0.81)
Negative indicators: worst, waste, boring.
Bogus: tv, alan, sun, brothers, develop, cut, male, higher, professor, attack

The final cross-validation RMSE while using 2000 features is 1.23.
TODO: Results for higher feature values.

### 3.2 L1 Loss function using stochastic gradient

The stochastic gradient converges at 0.91 after about 500 iterations with a feature size of 7000. Figure 1 shows the error vs. iteration plot.

*Strongest positives*: better, stars, agree
*Strongest negatives*: ashamed, worst, waste
Figure 1: Convergence Rate of Stochastic Gradient with feature size = 7000