Ratings Prediction Using Linear Regression on Text Reviews

Behavioral Data Mining, Assignment 2, Spring 2012

Eric Battenberg

February 24, 2012

1 Introduction

In this assignment, we use linear regression to predict book review rating scores (e.g. 1–5) using only the text from the reviews. The basic approach used was linear regression with a mean squared error cost function and either L2 or L1 regularization on the regression weights, as shown in eqs. 1,2.

$$\min_{\vec{w}} \frac{1}{N} \sum_{n=1}^{N} (y_n - \vec{w}^\intercal \vec{x}_n)^2 + \frac{\lambda}{2} \|\vec{w}\|_2^2 \tag{1}$$

$$\min_{\vec{w}} \frac{1}{N} \sum_{n=1}^{N} (y_n - \vec{w}^\intercal \vec{x}_n)^2 + \lambda \|\vec{w}\|_1 \tag{2}$$

As input features, we used unigram and bigram counts extracted from the text of each review. We also experimented with binary (term existence, not counts) unigram and bigram features.

Optimization of (1) and (2) was done using stochastic gradient descent augmented by a Quasi-Newton (Hessian estimating) algorithm.

2 Optimization Algorithm

Because (1) could not be solved exactly due to problems with matrix singularity and memory contraints, we used stochastic gradient descent to minimize both objective functions in (1),(2). In order to speed up learning, we used the "Corrected SGD-QN" algorithm [1], which estimates a diagonal approximation of the Hessian every few iterations. This algorithm corrects some theoretical flaws in the algorithm presented in the original "Stochastic Gradient Descent Quasi-Newton" (SGD-QN) algorithm [2].

In addition to computing a Hessian estimate, this algorithm ignores the regularization term when performing most updates and only performs a large regularization-term-only gradient update when updating the Hessian estimate. This aspect of the algorithm saves a large amount of computation when input data is sparse because the weight vector $\vec{w}$ is typically dense.

3 Setup

We partitioned 960,000 book reviews [3] into : 60% training, 30% testing, and 10% validation. Before training, it took about 15 minutes each (30 minutes total) to extract the unigram and bigram counts from the reviews and to build the sparse matrices where the features were stored.

To attempt to alleviate the significant class imbalance in the data (about 65% of the ratings were a 5), we introduce a non-uniform weighting to the error associated with each class. The weighting used is inversely proportional to the number of reviews in each class. The hope was that this weighting would increase the overall Area Under the Curve (AUC) of the ROC plot results, but this was not the case, as is shown in Section 4.

Training on unigram+bigram features for 100 sweeps through the data took approximately 2 minutes per parameter configuration.

4 Results

We trained each regression model for 100 epochs (complete sweeps through the training data). For stochastic gradient descent, the batch size was 1000 data points, and each epoch was composed of 576 batches of data. The initial learning rate that scales each gradient-based update was 0.01. The learning rate for each component of $\vec{w}$ is adjusted by the Hessian estimate and decay parameter from the SGD-QN algorithm. The Hessian estimate and learning rate are updated every 16 batches.

Training for 100 epochs using 12141 unigrams and 8772 bigrams as features takes approximately 2 minutes. We consider this a very reasonable amount of time in which to train such a large model. We attribute this favorable performance to the use of sparse matrix – compressed sparse row (CSR) format – operations which allow us to hold the entire training set in memory quite easily. In addition, CSR greatly speeds up the matrix-vector multiplications that are required during gradient calculations. The SGD-QN algorithm also allows the training to converge in much fewer iterations and further speeds up the training by allowing the regularization update to only occur every 16 batches. Our implementation was written in Python using the Numpy/Scipy modules for linear algebra operations.

As shown in Figure 1, AUC performance is worse across the board for models trained using the class weighted cost function mentioned in the previous section. Our best explanation for this is that giving additional weight to a class with such few training examples hurt the ability of the training to generalize to new data.

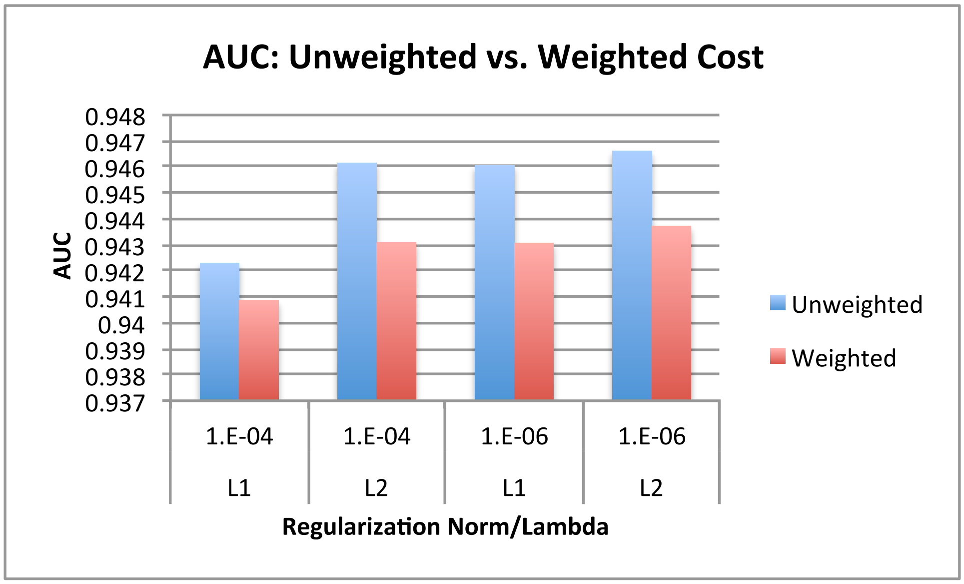


Figure 1: *AUC for models trained with and without class weighting.*

Figure 2 demonstrates the benefit using unigrams+bigrams as features. Unigrams alone perform nearly as well as unigrams+bigrams. Bigrams alone perform significantly worse, and we guess this is due to the general sparsity of bigram features. Considering a much larger set of bigrams could help, although limiting textual features to two-word phrases is quite a linguistic restriction.

Last, we show the AUC/1% Lift results for the unigram+bigram feature set trained with an unweighted cost function for various regularization parameter configurations. Figure 3 shows a slight edge for the L2 regularizer, especially at larger lambda values. In general, the best results were obtained using smaller values of lambda.

For the best performing model, the top positive and negative terms along with their associated weights are shown in Table 1. The terms are very illustrative of positive or negative reviews (though the fact that the model is sensitive to the presence of the term "five stars" could be considered cheating). Also, it's interesting to see that a term not easily associated with positive reviews like "negative reviews" is a strong indicator of a positive review. It seems that many reviewers tend to express their disgust with other negative evaluations of their favorite books.

Late breaking results: A few last minute runs using a binarized version of the data, where '1' denotes the existence of a unigram/bigram in the review, show a slight but promosing increase in performance over the results presented in Figure 3. We did not have to time to thorougly test and plot this observation, but we though it was worth sharing. The best performing model when using full uni/bigram counts achieved an AUC of 0.9550 and a 1% Lift of 50.11. In

Positive Term	**Weight**	**Negative Term**	**Weight**
__BIAS__	4.14813	disappointing	-0.971764
five stars	0.292859	waste	-0.82052
couldnt put	0.277786	disappointment	-0.75387
awesome	0.246602	dont buy	-0.662329
excellent	0.241283	useless	-0.609288
negative reviews	0.228613	garbage	-0.603105
didnt want	0.222544	boring	-0.582366
best book	0.21812	worst book	-0.5674
outstanding	0.213361	poorly	-0.55849
invaluable	0.212584	misleading	-0.504742
bad reviews	0.205332	worst	-0.490587
cant wait	0.20531	trash	-0.487643
masterpiece	0.202253	awful	-0.464711
great book	0.19924	disappointed	-0.459222
required reading	0.196785	nothing new	-0.458999
fantastic	0.195825	unreadable	-0.440433
even better	0.193302	worthless	-0.431778
5 stars	0.187752	outdated	-0.429157
superb	0.187638	dont waste	-0.422328
important book	0.184593	lame	-0.412592
hilarious	0.183424	better books	-0.410663
thank	0.177014	one star	-0.407822
well worth	0.176622	drivel	-0.389606
loved	0.175422	terrible	-0.3815
pleased	0.174067	skip	-0.372932
dont let	0.171718	poorly written	-0.367077
refreshing	0.171304	two stars	-0.3636
bravo	0.170913	mediocre	-0.361224
never boring	0.169171	dissapointed	-0.360859
gem	0.168176	zero	-0.356757
dont miss	0.166306	lacks	-0.356736
waste time	0.166169	disgusting	-0.354375
fabulous	0.165705	beware	-0.3539
nothing short	0.165575	tedious	-0.351644
funniest	0.16491	horrible	-0.349262
amazing	0.164564	shallow	-0.342046
favorites	0.163912	stay away	-0.340802
rocks	0.163831	zero stars	-0.339163
really good	0.161623	pathetic	-0.336543
extremely helpful	0.159407	unrealistic	-0.332609
nothing else	0.159108	sorry	-0.3277
book 5	0.157704	dull	-0.324211

Table 1: *Top positive and negative terms for the model trained using L1 cost function and a lambda of 1E-5.*

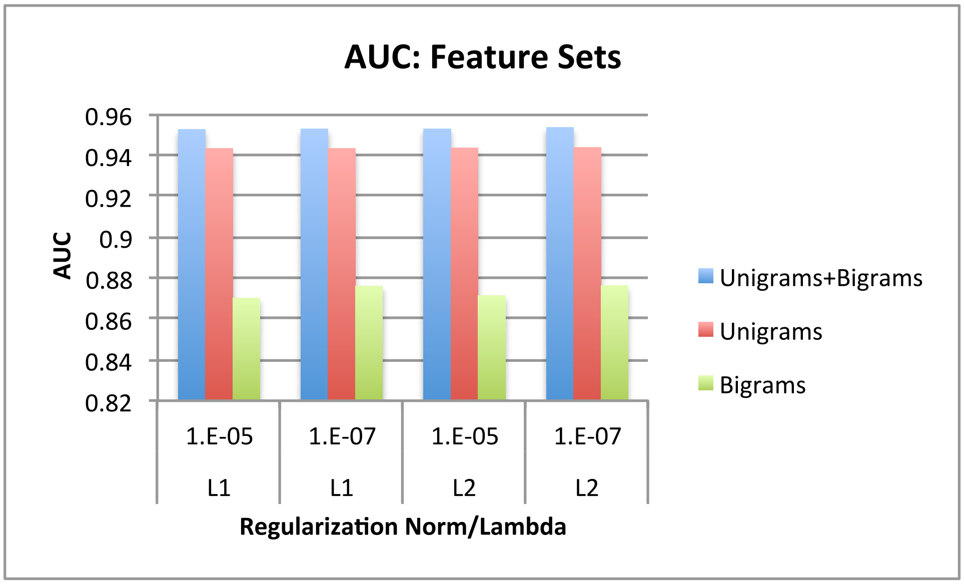
four quick runs using binary features, we found a model that achieved an AUC of 0.9558 and a 1% Lift of 52.39.

References

[1] A. Bordes, L. Bottou, P. Gallinari, J. Chang, and S. Smith, "Erratum: SGDQN is less careful than expected," *The Journal of Machine Learning Research*, vol. 11, pp. 2229–2240, 2010.

[2] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent," *The Journal of Machine Learning Research*, vol. 10, pp. 1737–1754, 2009.

[3] M. Dredze. Multi-Domain Sentiment Dataset (version 2.0). [Online]. Available: http://www.cs.jhu.edu/~mdredze/datasets/sentiment/



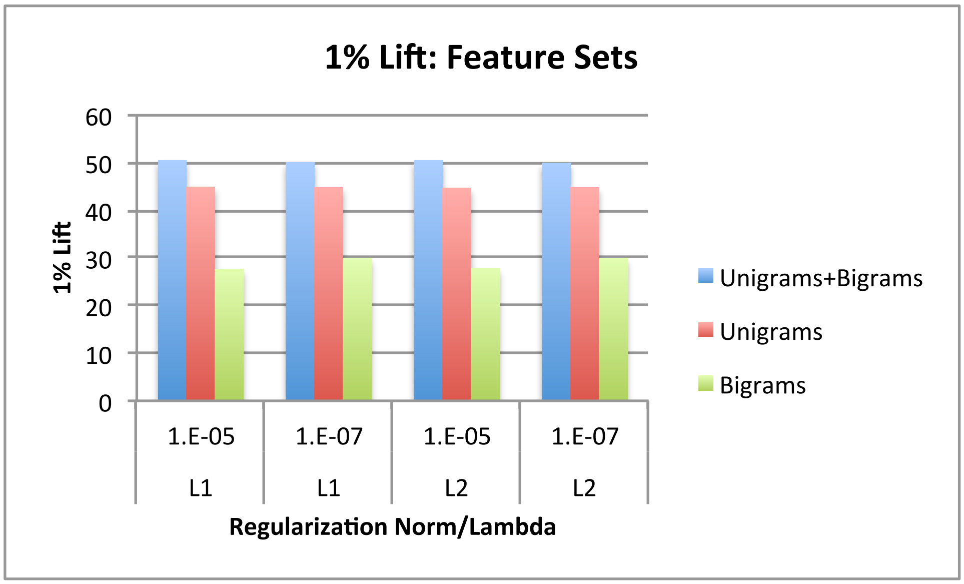
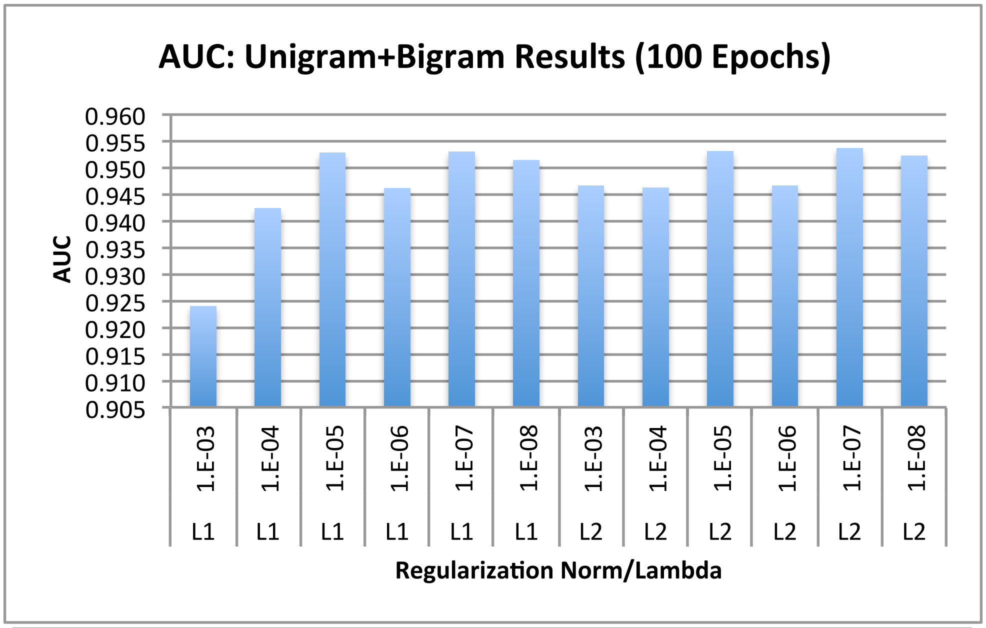


Figure 2: *Performance comparison for unigram/bigram combinations. Unweighted cost function.*



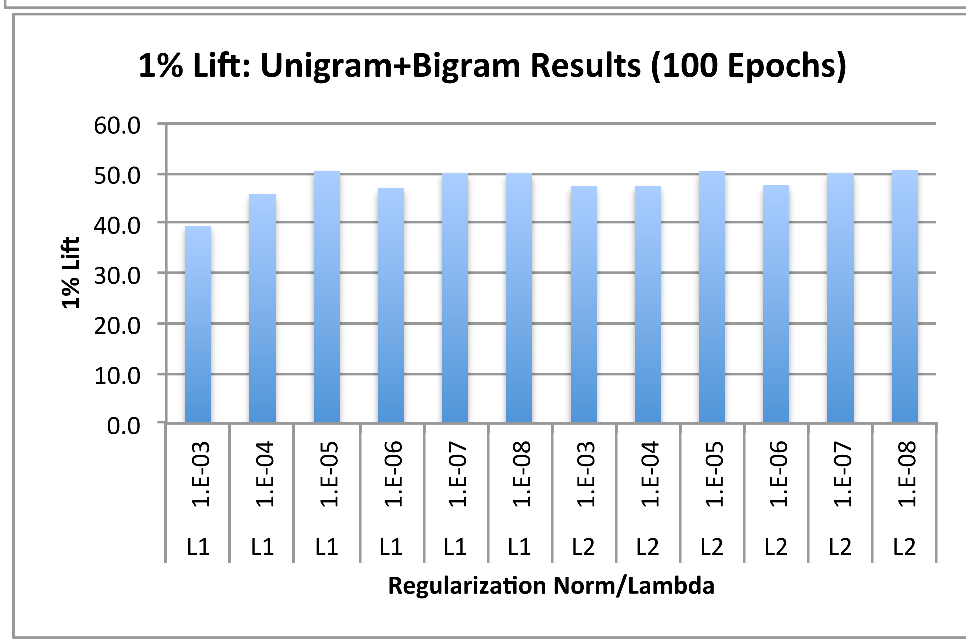


Figure 3: *Top: Area under ROC curve (AUC) for various regularizer settings. Bottom: 1% Lift of ROC curve.*