CIS 520 MACHINE LEARNING FINAL PROJECT REPORT

By-All-Means Classifier
Megha Agrawal, Pranshu Sharma, Varsha Shankar

INTRODUCTION

The aim of the project was to lower the root mean squared error for the prediction on the star rating of an Amazon review. We took a supervised learning approach to this problem, with our best classifier being one that predicts the average output of six classifiers (three methods with two separate training data). Two of our three methods were Logistic Regression and Support Vector Machine implementations of Liblinear library i.e. solver 7 and solver 3 respectively. The third method was Naive Bayes henceforth referred to as NB.

PRE-PROCESSING OF DATA

- Downloaded the porter stemmer implementation from [1] (we had to modify this to allow a corner case).
- Stemmed all unigram and bigram vocabulary. The bigram was split into its two components, stemmed and rejoined.
- Calculated the average count of each word, removed those that were below a cutoff. The intuition was that words that were not used often enough would not provide useful information.
- We removed outliers after stacking by finding out the set of examples that all the classifiers were getting wrong by a margin of more than 1. This number was around 6000 with just bigrams, and around 2000 with bigrams and unigrams. (Both considered title as well) We removed these examples, and then retrained the classifiers on the new dataset. This improved the test error from 0.96 to 0.89

TECHNIQUE THAT WORKED BEST

The best RMSE that we achieved was 0.8917. The sequence of steps followed were :

1. We used Porter Stemmer on bigram data as well as unigram data separately to reduce the number of features in each by removing repetition in bigram and unigram vocabularies (as explained in the pre-processing steps).
2. We then remapped the vocabulary of the examples to a new vocabulary that consisted of the stemmed words.
3. The six classification techniques we used were :
    a) Bigrams and Titles - NB
    b) Bigrams and Titles – Liblinear(s=3)
    c) Bigrams and Titles – Liblinear(s=7)
    d) Unigrams and Titles - NB
    e) Unigrams and Titles - Liblinear(s=3)
    f) Unigrams and Titles - Liblinear(s=7)
4. We used ‘text’ and ‘title’ as the features to start with and then filtered them out on the basis of a frequency cutoff. The features whose frequency of occurrence in the training sample fell below the mean frequency of all features were eliminated.
5. We used 11-fold cross validation (each fold corresponding to one category of reviews) to build 6 separate models at each stage of cross validation and aggregate their predicted rating labels into a matrix such that each example had 6 columns by the end of the 11-fold cross validation. Each column corresponded to the predicted rating by a different classifier (listed above in 3).
6. Instead of using the integer rating that each classifier estimated, we weighted the ratings on the basis of the classifier’s confidence and stored these values in our stacked matrix.
7. We used the stacked prediction matrix of the training data to remove outliers. The procedure we followed was to find the intersection of those examples where all 6 classifiers went wrong in their predicted ratings by more than 1.
8. The stacked training matrix that we now had consisted of ‘inlier’ examples only. We created sparse matrix of this training set. On this training set, we removed insignificant features by using frequency cut-off. Our

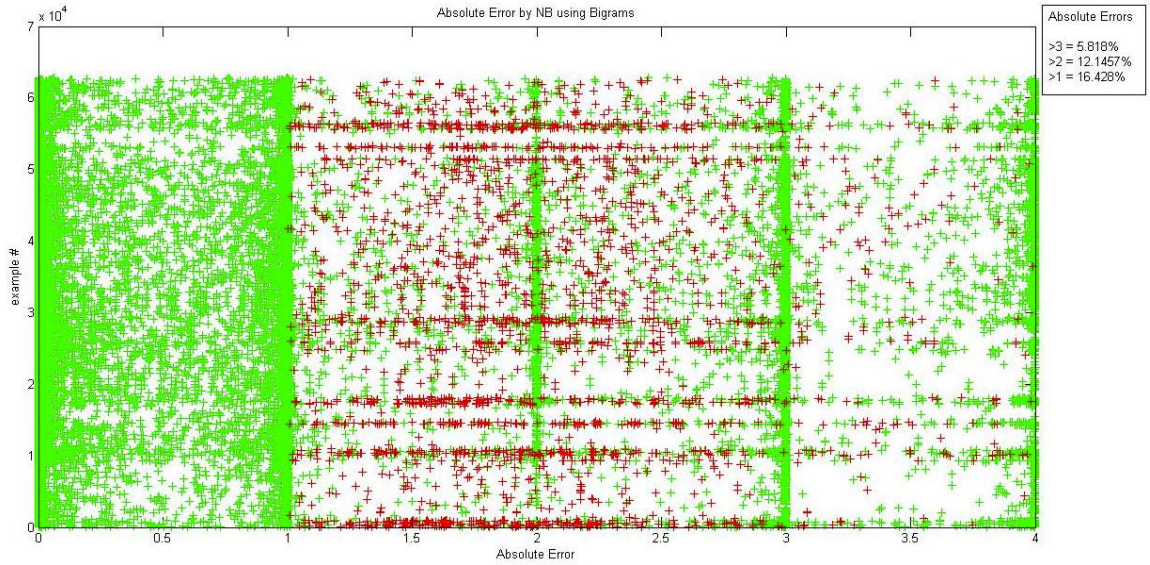
reasoning behind this was that features i.e. word counts that appeared very rarely across all the examples were unlikely to affect the overall rating of the examples.

9. We trained the same six models with the whole stacked training matrix.
10. We used the inbuilt cross validation in Liblinear to decide the cost parameters for solvers 7 and 3.
11. With these newly trained six models, we predicted 6 possible ratings for each of the examples in the test data.
12. The final prediction for each example was an average of the six outputs (which in hindsight seems a little stupid. Why didn't we use a weighted average!!!)

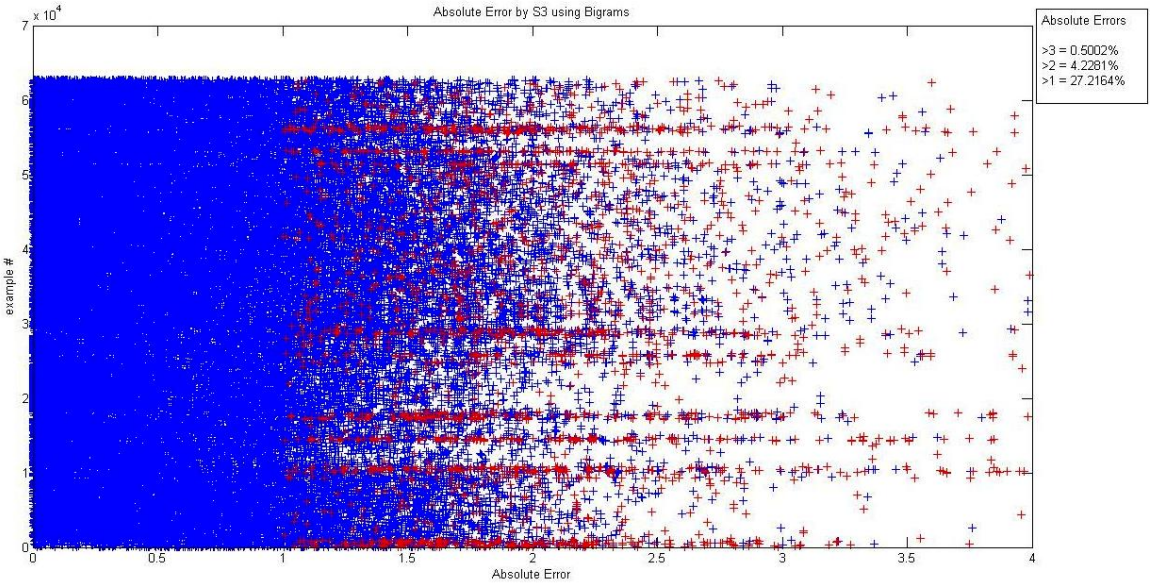
FANCY VISUALIZATION

For all graphs, the points in red are the data-points (outliers) that all 6 of our classifiers misclassified by an error margin greater than 1. There were around 2400 data-points that proved to be outliers. Removing these outliers improved our RMSE from 0.96 to 0.89.

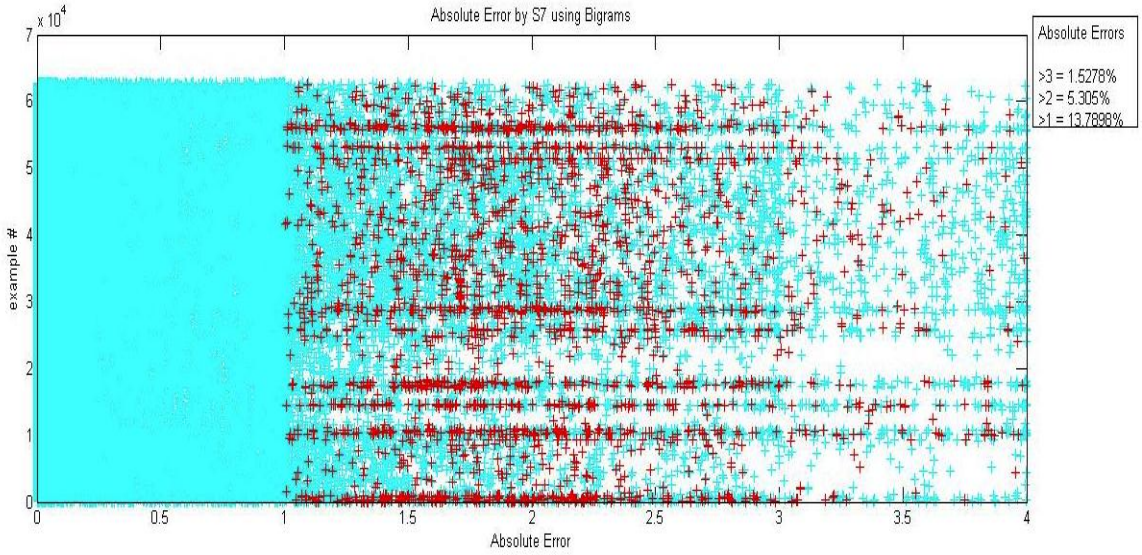
The figure below shows the absolute error between the true rating and the predicted rating for each example by an NB classifier trained on bigram word count and title word count.



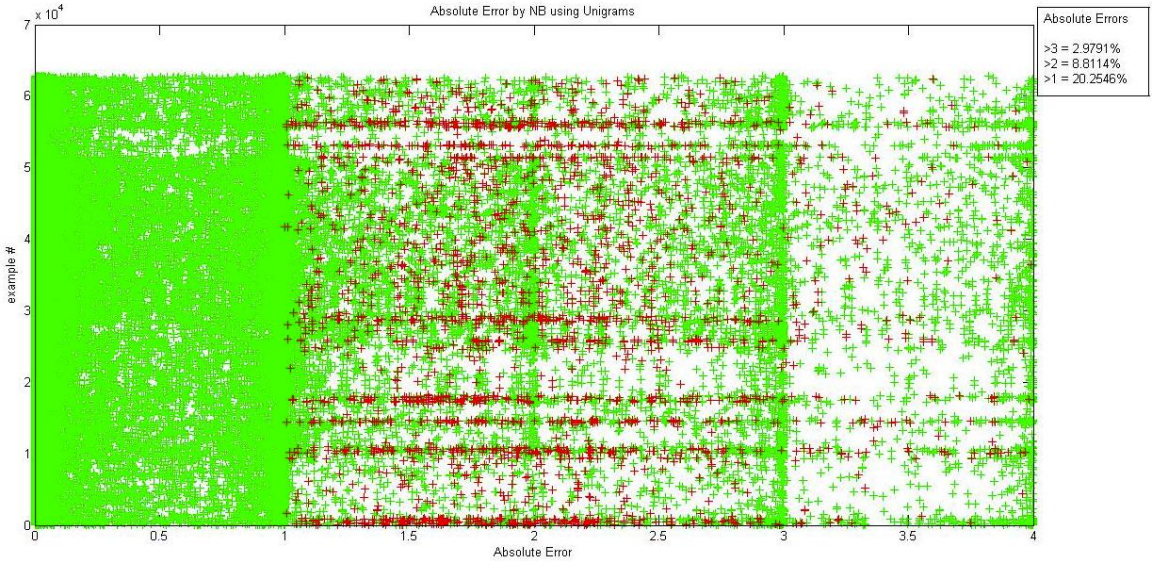
The figure below shows the absolute error between the true rating and the predicted rating for each example by Liblinear (s=3) classifier trained on bigram word count and title word count.



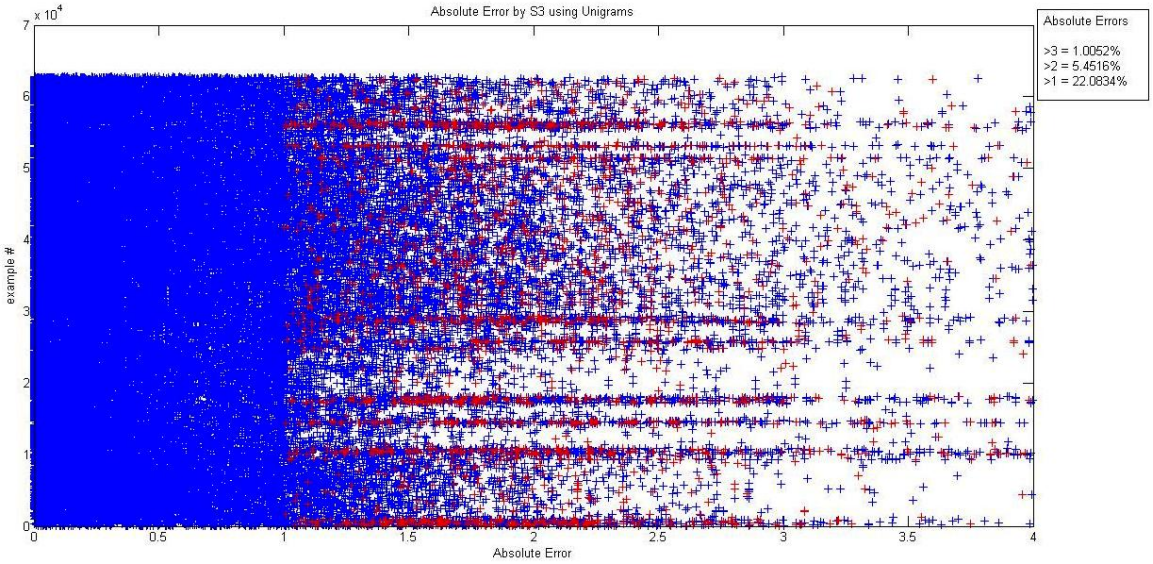
The figure below shows the absolute error between the true rating and the predicted rating for each example by Liblinear(s=7) classifier trained on bigram word count and title word count.



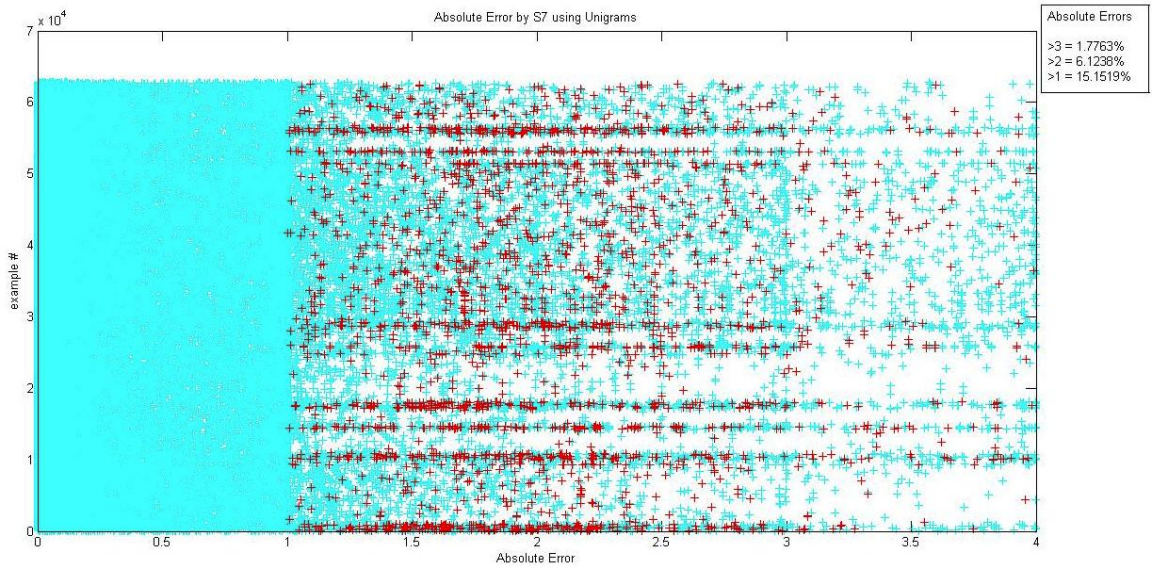
The figure below shows the absolute error between the true rating and the predicted rating for each example by an NB classifier trained on unigram word count and title word count.



The figure below shows the absolute error between the true rating and the predicted rating for each example by Liblinear(s=3) classifier trained on unigram word count and title word count.



The figure below shows the absolute error between the true rating and the predicted rating for each example by Liblinear(s=7) classifier trained on unigram word count and title word count.



ANALYSIS

In the Naïve Bayes graphs, the density of data-points with error margins less than 1 is much lower than for s7 and s3. We can see this increase in density rather clearly as we go from Naïve Bayes to s3 to s7. This indicates that precision of the classifiers increases in the same order. Similarly, the density of data-points in the region with error margins greater than 1 decreases in the same order. This shows that s7 makes fewer mistakes and is more confident about its predictions (since the density of higher error margin datapoints is low) than s3 and Naïve Bayes.

THINGS THAT DIDN'T WORK

1. **TF-IDF**: We replaced word counts for unigrams with tf-idf scores for each, and trained a Naive Bayes classifier, without any preprocessing. This gave rather bad results (RMSE = 1.55). Also, the Naive Bayes we used to train was the one provided, that took a logical matrix as input. Thus, the only way the input to Naive Bayes differed with tf-idf scores as features was that words that appeared in each document were now

given a score of zero (as their idf was zero.) So a probable cause for tf-idf failing rather miserably could be that these words (that appear in each example) were in fact more important than imagined.

2. Tried **stopwords** removal - We pruned a stopwords list obtained from [2] to remove all words that provided useful information (for example, words conveying negation). Our stopword list consisted of articles, possessive pronouns and prepositions. This didn't work too well, xval error increased for porter stemmed unigrams with stopwords removed trained on regular Naive Bayes, as compared to just porter stemmed unigrams trained on the same. We were very surprised by this rather counter-intuitive result and still have not been able to find an explanation for the same.
3. **Kernelized SVM** with random sampling of 1000 training samples - We grouped the training samples into four categories based on their ratings (1,2,4,5) and calculated a hypothetical 'centroid' by averaging feature values across all the samples in that category. From these four clusters, we sampled 250 data-points based on their Euclidean distance to the respective 'centroid'. These four data-points are ideally *most representative* of that category.
   We used this reduced set of training examples as our input to kernelized SVM with an intersection kernel. Using this method alone, our RMSE on the test set was 1.46. The performance of this method was similar to NB exposed to all 62000 training examples. We think this occurred because of two reasons viz.
   - The number of training samples we picked i.e. 250 to represent each rating category was not descriptive enough in terms of variety and volume. However, anything above this number was slowing down creation of the kernel and beyond a limit, not allowing one to be created.
   - We picked 250 samples from each category on the basis of SSD to the averaged centroid. This is perhaps not the best way to decide on the most representative examples. A better approach would have been to find out which features correlated most with the rating and picking those examples in the category that had the largest values for the most correlated features.
4. **KNN** with random sampling of preprocessed 1000 training samples - We generated the 1000 random training samples as explained above and plugged it into:
   - A 5-NN model, with ties broken according to the consensus rule.
   - A 5-NN model, with ties broken according to the nearest neighbor rule.
   - A 10-NN classifier, with ties broken according to the consensus rule

   The cross validation error for all three was around 1.5. KNN typically took longer time to execute than the other models.

CONCLUSION

We found that stacking with NB, Liblinear s3 and s7 worked best for us. A lot of ideas that intuitive made sense such as tf-idf and stopwords removal did not work as well as we thought.

REFERENCES

[1] http://tartarus.org/martin/PorterStemmer/matlab.txt
[2] http://www.lextek.com/manuals/onix/stopwords1.html