CIS 520 Final Project Report: Team Kernel Sanders

Brendan Callahan
`bcal@seas.upenn.edu`

Patrick Crutchley
`pcrutchl@psych.upenn.edu`

Michael Woods
`micwoods@mail.med.upenn.edu`

December 10, 2013

1 Background

The competition this year focused on developing a method of sentiment analysis for user reviews of restaurants and other businesses from a supplied Yelp dataset. As participants, the task laid before us was conceptually simple: given a users text review of a business, plus some metadata (outlined below), predict the corresponding rating, a discrete score between one and five stars.

The dataset itself consisted of 25,000 training reviews, 5000 quiz reviews (no labels provided), a vocabulary lexicon of 56,835 words, and associated metadata for each review. Each review was presented as a "bag of words", whereby each review was decomposed into a vector with entries denoting the presence or absence of the i-th word in the vocabulary lexicon, and if present, how many times that word appeared. The lexicon itself was constructed from the tokenized text of all training and quiz examples, with limited punctuation removed and all words normalized to lowercase. The metadata of each review consisted of a unique user ID, business ID, "cool" / "funny" / "useful" tags, and the full text of the review as a list of tokens in sequential order without any preprocessing applied.

In order to judge the effectiveness of the entries, the final assessment of predictor performance consisted of a dataset of 10,000 reviews, with corresponding labels that the final, trained learners would be asked to predict. The top teams with the learners that produced the lowest RMSE on the test dataset would be declared the winners.

2 Tools Used

MATLAB , specifically the Statistics toolbox was used to create the bulk of our predictive engine. Outside of the MATLAB standard library, we leveraged the LIBLINEAR MATLAB package as the underpinning for the implementation of most of our learners. This was due to LIBLINEAR 's excellent support for working with large matrices, specifically sparse ones. Outside of MATLAB , we used Python to process the full text of the reviews in order to generate bigrams. The decision to use Python was made, due in large part, to the cumbersome nature of text processing in MATLAB , and the availability of the SciPy package to read and write `.mat` files. Git, along with Github, was used for version control.

3 Structure

Based on the results of the Netflix prize, and the intuition that the combination of multiple opinions are "better" than one, from the outset we believed that an ensemble of learners would most likely yield the best results. As far as metrics go, "better" in the case of this competition meant creating a final predictor that achieved the lowest possible RMSE during testing, while avoiding the effect of overfitting when the same code was run against out-of-sample test reviews.

A number of methods were employed to combine the output ratings from multiple predictors. Specifically, we implemented the following methods:

1. **Weighted average, with constant weights.** The ratings were simply averaged together, with each prediction having the same importance

2. **Weighted average, with weights determined by linear regression.** Predicted ratings were first generated from each learner using the training dataset, then a linear regression was run with the predicted ratings serving as the observation matrix, $\mathbf{X}$, and the actual ratings as the label vector, $\mathbf{Y}$. The resulting weight vector, $\mathbf{w}$, when used to generate a weighted average prediction, with the i-th entry of w scaling the vector of predicted ratings produced by the i-th learner.

3. **Weighted average with weights determined by stepwise regression.** Like weighted average with linear regression above, except stepwise regression was used in place of linear regression.

4. **Simple majority vote.** As the name implies, a simple majority vote was taken for the i-th prediction. In the event each learner produced a different prediction for a given review, the prediction of the first learner was used.

4 Testing

In order to test our learners, we used a simple N-fold cross validation scheme, where the training dataset was divided into N as-close-to-equally sized "folds" as possible. All reviews within the i-th fold were assigned an ordinal number i, then shuffled. Of the N folds, $N-1$ constituted the training set of reviews the learners were trained on, and the remaining held out reviews were used to gauge the accuracy of our predictions. Typically, cross validation was run with 10 folds; this produced a training set of 22,500 reviews and a test set of 2,500 reviews. $N = 10$ was chosen, as it provided a sufficiently large number of runs to calculate an accurate average root mean square error (RMSE).

5 Methods Employed

What follows is a list of all machine learning methods employed during the course of the competition by our team, with a short analysis of each method, its relative effectiveness, and any difficulties we experienced.

Each learner that was implemented as MATLAB package was required to define three functions, train, predict, and test, that defined a simple interface for the training and cross validation code to interact with the learners in a simple and consistent way.

The learners are referenced by the names of the packages, as they existed in our code. They are short, and fairly self-explanatory. They are divided into two categories below: Successful and Unsuccessful. As the names imply, Successful contains the learners used to construct our final ensemble learner, whereas learners labelled Unsuccessful either did not work, took too long to run, or did not yield useful or especially accurate predictions.

5.1 Successful Component Learners

5.1.1 `counts_logit_reg`

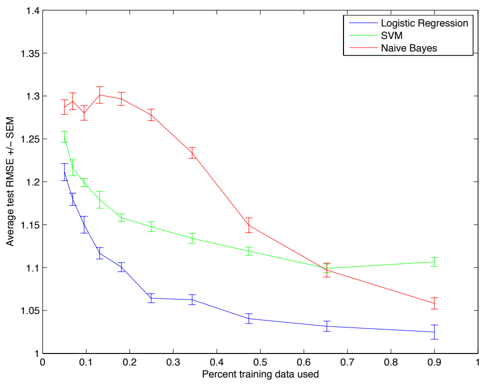
A multiclass, L2-regularized logistic regression implementation from the LIBLINEAR package was trained directly on the bag-of-word counts for each review. By itself, cross validation (N=10) revealed that logistic regression achieved an average RMSE of 1.019, and an average predictive accuracy of 49.38%. The counts logistic regression learner proved to be one of the foundational learners used in the final learner ensemble, due to its relatively high accuracy compared to other methods tested. L2 regularization was found to have best performance through 10-fold cross validation accuracy on the training set.

5.1.2 `counts_svm`

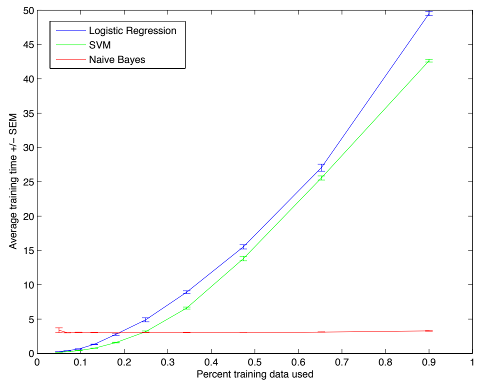
A multiclass, L2-regularized, L2 loss SVM implementation from LIBLINEAR was, like `counts_logit_reg`, trained on the bag-of-words counts for each review. Again, when used in isolation, cross validation (N=10) produced an average RMSE of 1.095, and an average accuracy of 46.08%. The SVM parameters used in the final learner proved to be somewhat under-regularized, i.e., providing an over-fit model using the entire training set. See Figure 1a; the RMSE on a test set of 10% of the data showed an uptick using somewhere between 65% and 90% of the training data. Given the other learners in the final ensemble, however, this overfitting was not fatal; `counts_svm` still contributed positively toward the final ensemble.

5.1.3 `nb`

A Naïve Bayes classifier implemented using the MATLAB `NaiveBayes` class using a multinomial distribution for training. The multinomial distribution was chosen, as its a bag-of-tokens model for the underlying data distribution mirrored how the data was stored for each review. With cross validation (N=10), the Naïve Bayes learner yielded an average RMSE of 1.057 and an average accuracy of 48.69%. A positive aspect of the Naïve Bayes classifier, at least the MATLAB implementation thereof, was a relatively flat train-time-versus-training-set-size relationship; see Figure 1b. Both the logistic regression and SVM learners displayed an approximately-quadratic growth rate in training time duration with number of observations trained. The Naïve Bayes implementation was much more constant. Even using the entire training set, class `nb` provided fast results. `nb`, along with `counts_svm`, and `counts_logit_reg`, composed our final ensemble learner.



(a) RMSE performance of learners versus a 2500-sample holdout set versus size of training set. (Total training set size is 25,000, so test set size is a constant 0.1 percentage.)



(b) Training time of learners (in sec) versus a 2500-sample holdout set versus size of training set. (Total training set size is 25,000, so test set size is a constant 0.1 percentage.)

Figure 1: Behavior of learners with increased training set size

5.2 Unsuccessful Learners

5.2.1 `eigenwords_logit_reg`

Bigram matrices were generated from the combined training and quiz sets (using the complete text as found in the supplied metadata). We performed a singular value decomposition (SVD) on the bigram matrices, generating reduced-dimension left and right singular vectors (**U** and **V**, respectively) for each word in the vocabulary. Predictors for each document were the average position (centroid) of the document's words in the reduced-dimension space. This method was patterned off of Foster, Liberman, and Stine (unpublished)[1].

As revealed through cross-validation, prediction accuracy was, on average, was less than 30%. As such, the performance of `eigenwords_logit_reg` was less than originally anticipated. We believe that the contrast between the performance found by Foster et al and us could be the result of a regression problem versus a classification problem, respectively, or a different performance metric: R^2 versus RMSE, respectively.

[1] "Featurizing Text: Converting Text into Predictors for Regression Analysis". Dean P. Foster, Mark Liberman, and Robert A. Stine. Unpublished.

5.2.2 `multi_mira_perceptron`

A two-pass, multiclass perceptron/MIRA implementation derived from UC Berkeley CS 188[2] and CIS 520 lecture notes. The predictive accuracy for two passes was 45%; for three passes, almost 49%. Ultimately, this learner was not used due to the amount of time necessary to train it, since for every review, the dot product of two large vectors needed to be computed per rating. Each vector was the size of the vocabulary lexicon, so each per review, $5 \times 56,835$ multiplications needed to take place, along with $5 \times 56,835$ resulting addition operations. For the training dataset in cross-validation ($N = 10$), roughly $10 \times (2 \times 22,500 \times (5 \times 56,835)^2)$ operations needed to take place total.

5.2.3 `kmeans`

The top 75 principal components were selected using MATLAB built-in `svds`, and then fed into `kmeans` ($K = 100$). The model then consisted of the centroids for the cluster, and the predicted rating for each cluster. The predicted rating of each cluster is determined by using the mean rating of all training data points that are members of that cluster. Predictions are done by taking the minimum euclidean distance between each test point and the clusters, and using the prediction for the minimally distanced cluster. Ultimately the distance metric used for the predictions needed major improving, primarily by using actual feature selection in order to do a like-for-like comparison of the test point to the cluster centroid. The mean RMSE during cross-validation of this classifier was on-par with `nb`/`counts_svm`/counts_logit, but the actual RMSE values swung a bit more wildly. Various K values were tried, but none significantly improved the results.

5.2.4 `business_user_classifier2`

Unique index values were created for each business and user. These were then used to find the average rating value across all users for that business/user. To fill in missing values (businesses and users that were not in the training data), we used the average rating across all training data points. This method performed well in cross-validation, especially when paired with `counts_logit_reg`/`counts_svm`/`nb`, it brought our mean RMSE down to ~ 0.87. However the approach sufferred from overfitting against the training data, as the performance against the quiz set dropped to ~ 1.05.

5.2.5 `funny_cool_useful`

This was a very simple 3-feature learner using the same logistic regression values as in `counts_logit_reg`. On its own, the cross-validation performance was around 1.5, but when paired with `counts_logit_reg`, `counts_svm`, and `nb`, it offered a small but consistent boost to the cross-validation score. Ultimately we deferred testing it on quiz in favor of methods that may have offered larger improvements to the mean RMSE.

6 Challenges

6.1 Dimensionality

Due to the huge number of dimensions per review in the dataset, dimensionality reduction of some sort was needed in order to train certain classes of learners. For example, MATLAB s built-in `kmeans` function was entirely unusable since we speculate within the function, it converts any sparse matrix passed into it into a full matrix. Any manipulations of the full matrix representation of the training dataset became all but impossible with a commodity piece of hardware like a MacBook. Moreover, we could not use MATLAB s built-in functions for principal component analysis like `pca` or `princomp`, as performing mean-centering on the sparse training data matrix converted it to a full matrix, which again, made it intractable to work with. One potential workaround we utilized to a varying degree of success involved simply performing SVD on a sparse matrix itself without mean centering, and taking the top $k + 1$ singular vectors. The first singular vector was discarded, and the rest were used as the top k principal components. ***Maybe some analysis of why this did not perform too well? ***

[2] `http://inst.eecs.berkeley.edu/~cs188/fa09/projects/classification/classification.html`

6.2 Final Learner

The final learner used to generate predictions for the assessment test set consisted of `counts_logit_reg`, `nb`, `counts_svm`. The resulting predicted rating from the three learners were combined using a weighted average with the following weights: [0.4170.3680.172]. The weights were determined experimentally through cross-validation ($N = 10$), and observing that that weights produced via linear regression consistently produced the lowest, or second lowest RMSE for each round of cross-validation.

To meet the test set speed performance requirements with MATLAB 's built-in `NaiveBayes` functionality, a monkey-patched version was created which created that stored the output of `cell2mat(nb_model.Params)` in `init_models`. We then passed that stored value into the monkey-patched version of `predict.m`, which called into the monkey-patched version of `NaiveBayes.m`. This one change allowed us to boost our speed from ~ 8 iterations/sec to ~ 140 iterations/sec.

Our performance against the quiz set was as follows: Learners Mean RMSE `counts_logit_reg` 1.0314 `counts_logit_reg`, `nb` 0.9723 `counts_logit_reg`, `nb`, `counts_svm` 0.9548 `counts_logit_reg`, `nb`, `counts_svm`, kmeans 0.9446 `counts_logit_reg`, `nb`, `counts_svm`, business_user_classifier2 1.0413

*** Brendan and/or Patrick *** If anyone has numbers for quiz performance, etc. saved we could insert a table here and talk more about performance metrics ***

Conclusion *** What are peoples thoughts here? How could we improve our results, etc? *** We performed reasonably well with 3 relatively simple classifiers: naive bayes, logistic regression, and support vector machines. Our performance was also consistent across cross-validation, against the quiz set, and against the test set, which shows we did not have any trouble with overfitting. Improved feature selection was one of the main areas that could have helped our results. It would have allowed us to run kmeans and get reasonable predictions at least. Additionally, creating some sort of similarity metric for comparing businesses/users would have been useful for filling in the businesses/users we had no prior knowledge of. Improving the n-gram creation to skip common stop words may have improved the performance of the eigenwords-based approaches.