Yelp Dataset Challenge: Review Rating Prediction

Nabiha Asghar NASGHAR@UWATERLOO.CA
University of Waterloo, 200 University Avenue West, Waterloo, ON N2L3G1 Canada

Abstract

Review websites, such as TripAdvisor and Yelp, allow users to post online reviews for various businesses, products and services, and have been recently shown to have a significant influence on consumer shopping behaviour. An online review typically consists of free-form text and a star rating out of 5. The problem of predicting a user's star rating for a product, given the user's text review for that product, is called Review Rating Prediction and has lately become a popular, albeit hard, problem in machine learning. In this paper, we treat Review Rating Prediction as a multi-class classification problem, and build sixteen different prediction models by combining four feature extraction methods, (i) unigrams, (ii) bigrams, (iii) trigrams and (iv) Latent Semantic Indexing, with four machine learning algorithms, (i) logistic regression, (ii) Naïve Bayes classification, (iii) perceptrons, and (iv) linear Support Vector Classification. We analyse the performance of each of these sixteen models to come up with the best model for predicting the ratings from reviews. We use the dataset provided by Yelp for training and testing the models.

1. Introduction

User reviews are an integral part of web services like TripAdvisor, Amazon, Epinions and Yelp, where users can post their opinions about businesses, products and services through reviews consisting of free-form text and a numeric star rating, usually out of 5. These online reviews function as the 'online word-of-mouth' (Dellarocas, 2003) and a criterion for consumers to choose between similar products. Studies (e.g. (Chen et al., 2003)) show that they have a significant impact on consumer purchase decisions as well as on product sales and business revenues. A user review typically looks like this:

> *Restaurant: XYZ, Kitchener, N2G4Z6, Canada*
> *Rating:* ●●●○○
> *I've been to XYZ a bunch of times. It's a decent place. Nice food, lots of variety! The place is really small though, so you almost never find a spot to sit and eat. The service is also slow at times.*

Figure 1. A Typical User Review: Free-form Text & a Star Rating

On famous websites like Amazon and Yelp, many products and businesses receive tens or hundreds of reviews, making it impossible for readers to read all of them. Generally, readers prefer to look at the star ratings only and ignore the text. However, the relationship between the text and the rating is not obvious, as illustrated in Figure 1. In particular, several questions may be asked: why *exactly* did this reviewer give the restaurant 3/5 stars? In addition to the quality of food, variety, size and service time, what other features of the restaurant did the user implicitly consider, and what was the relative importance given to each of them? How does this relationship change if we consider a different user's rating and text review?

The process of predicting this relationship for a generic user (but for a specific product/business) is called Review Rating Prediction. Concretely, given the set $S = \{(r_1, s_1), ..., (r_N, s_N)\}$ for a product P, where r_i is the i'th user's text review of P and s_i is the i'th user's numeric rating for P, the goal is to learn the best mapping from a word vector $\mathbf{r}$ to a numeric rating s. Review Rating Prediction is a useful problem to solve, because it can help us decide whether it is enough to look at the star ratings of a product and ignore its textual reviews. Moreover, some review websites allow users to write text reviews without specifying a star rating. In these cases, Review Rating Prediction comes in handy. However, it is a hard problem be-

cause two users who give a product the same rating, may have very different reasons for doing so. User A may give a restaurant 2/5 stars because it does not have free wifi and free parking, even though the food is good. User B may give the same restaurant a rating of 2/5 because he does not care about the wifi and parking, and thinks that the food is below average. Therefore, the main challenge in building a good predictor is to effectively extract useful features of the product from the text reviews and to then quantify their relative importance with respect to the rating.

In this paper, we treat Review Rating Prediction Problem as a multi-class classification problem in Machine Learning, where the class labels are the star ratings. We combine four feature extraction methods, unigrams, bigrams, trigrams and Latent Semantic Indexing, with four supervised learning algorithms, logistic regression, Naïve Bayes classification, perceptrons and linear support vector classification to build sixteen prediction models. We train and evaluate the performance of each of these models on the dataset provided by Yelp. The rest of the paper is organized as follows. Section 2 and 3 provide the details of the related work and the dataset. Section 4 describes all the feature extraction methods and supervised learning algorithms, and section 6 provides detailed results and analysis. We end with concluding remarks and future work.

2. Related Work

Most of the recent work related to review rating prediction relies on sentiment analysis to extract features from the review text. Qu *et al.* (2010) tackle this problem for Amazon.com reviews, by proposing a novel feature extraction method called bag-of-opinions, which extracts opinions (consisting of a root word, a modifier and/or a negation word) from the review corpus, computes their sentiment score, and predicts a review's rating by aggregating the scores of opinions present in that review and combining it with a domain-dependent unigrams model.

Leung *et al.* (2006) use a novel *relative frequency method* to create an opinion dictionary, in order to infer review ratings from the review text. This method estimates the strength of a word with respect to a certain sentiment class as the relative frequency of its occurence in that class. They integrate this inference technique with collaborative filtering algorithms and test their method on movie reviews from IMDb on a 2-point rating scale.

Fan and Khademi (2014) predict a restaurant's average star rating on Yelp from its reviews (note that this is business rating prediction, and is different from review rating prediction). They combine the unigrams model with feature engineering methods such as Parts-of-Speech tagging, and use linear regression, support vector regression and decision trees for prediction. Their dataset consists of 4243 restaurants and 35645 text reviews, and is much smaller than the one we use in this paper. Li *et al.* (2011) perform rating prediction for reviews on Epinions.com by extracting additional features of the reviewer and the product/business being reviewed. Ganu *et al.* (2009) propose a method to use the text of the reviews to improve recommendor systems, like the ones used by Netflix, which often rely solely on the structured metadata information of the product/business and the star ratings. Their method relies on machine learning, sentiment analysis techniques and natural language processing to classify sentences as positive, negative, neutral or conflict. It is shown, using restaurant reviews from Citysearch New York, that the review text is a better indicator of the sentiment of the review than the coarse star rating.

In this paper, we concern ourselves only with the semantic analysis of the review text and do not deal with the sentiment analysis.

3. Data Description

We use the dataset provided by Yelp as part of their Dataset Challenge 2014 (Dataset, 2014) for training and testing the prediction models. The dataset includes data from Phoenix, Las Vegas, Madison, Waterloo and Edinburgh, and contains information about 42,153 businesses, 320,002 business attributes, 31,617 check-in sets, 403,210 tips and 1,125,458 text reviews.

Concretely, the dataset consists of five files, one for each object type: business, review, user, check-in and tip. Each file consists of one json-object-per-line. Thus, a business is represented in the 'business.json' file as a json object which specifies the business ID, its name, location, stars, review count, opening hours, etc. A text review is a json object in the 'review.json' file, which specifies the business ID, user ID, stars (integer values between and including 1 and 5), review text, date and votes. The necessary data is contained in the business.json and review.json files, therefore we do not use the rest of the data.

The businesses described in the Yelp dataset belong to different categories, such as restaurants, shopping, hotels and travel, etc. The text reviews for different business categories may be very different. For example, a typical hotel review may contain the words/phrases 'fridge', 'television' and 'comfortable bed', but these

words would not occur in a restaurant review. Therefore, it is important to perform Review Rating Prediction for each business category independently. That is, the model training and testing for each category should be separate. Figure 2(a) shows the distribution of business categories in the dataset. Restaurants make up almost 34% of the 42,153 businesses. Moreover, 68.3% of the 1,125,458 text reviews are about restaurants, as shown by Figure 2(b). Therefore, in this paper, we restrict ourselves to Review Rating Prediction for restaurants only. Thus, the trimmed dataset that we use consists of 14,403 restaurants and 706,646 reviews. Figure 2(c) shows that the star ratings (out of 5) for the restaurant reviews are not uniformly distributed. About 66% of these reviews rate the corresponding restaurants very highly (at least 4 stars); the other classes are smaller.

4. Experimental Setup

In this paper, we build sixteen different prediction models, by combining each of four different feature extraction methods with each of four distinct supervised learning algorithms. In this section, we describe the preprocessing phase, the four feature extraction methods, the four supervised learning algorithms, and two performance evaluation metrics.

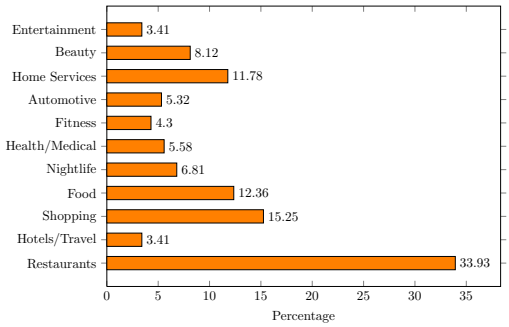
4.1. Preprocessing

We first write some basic Python scripts to separate the restaurants from the business.json file, and to separate the restaurant reviews from the review.json file. We then preprocess the text reviews as follows.

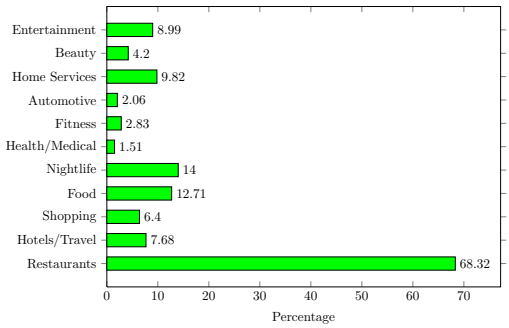
Yelp allows users to write text reviews in free form. This means that a user may excessively use capital letters and punctuation marks (to express his/her intense dislike, for example) and slang words within a review. Moreover, stop words, like 'the', 'that', 'is' etc, occur frequently across reviews and are not very useful. Therefore it is necessary to preprocess the reviews in order to extract meaningful content from each of them. To do this, we use standard Python libraries to remove capitalizations, stop words and punctuations.

4.2. Feature Extraction

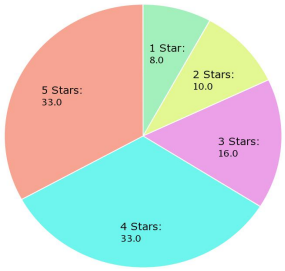
We use four methods to extract useful features from the review corpus and to build a feature vector for each review. Each of these methods relies on semantic analysis of the text.



(a) Distribution of Business Categories



(b) Distribution of Reviews



(c) Stars Distribution for Restaurant Reviews

Figure 2. Descriptive Stats: Yelp Businesses & Reviews

4.2.1. Unigrams

In the uni-grams model (also called the "bag of words" model), each unique word in the pre-processed review corpus is considered as a feature. Thus, building a feature vector for a review is straightforward. First, a dictionary of all the words occuring in the review corpus is created. Then a word-review matrix is constructed, where entry (i, j) is the frequency of occurence of word i in the j'th review. Finally, we apply the TF-IDF (Term Frequency - Inverse Document Frequency) weighting technique to this matrix to obtain the final feature matrix. This weighting technique assigns less weight to words that occur more frequently across reviews (e.g. "food") because they are generally not good distinguishers between any pair of reviews,

and a high weight to more rare words. Each column of this matrix is a feature vector of the corresponding review.

The size of the dictionary in this case, i.e. the total number of features, is 171,846.

4.2.2. Unigrams & Bigrams

The unigrams model is a widely used feature extraction method in natural language processing. It is quite simple to implement and in many cases, it gives surprisingly good results. However, its inherent drawback is its inability to capture relationships between two words (e.g. a word and its modifier, a word and its negation, etc), because it treats each word in isolation. To capture the effect of phrases such as 'tasty burger' and 'not delicious', we *add* bigrams to the unigrams model. Now, the dictionary additionally consists of all the 2-tuples of words (i.e. all pairs of consecutive words) occuring in the corpus of reviews. The matrix is computed as before; it has more rows now. As before, we apply TF-IDF weighting to this matrix so that less importance is given to common words and more importance is given to rare words.

The size of the dictionary (total number of features) is 7,612,422.

4.2.3. Unigrams, Bigrams & Trigrams

To capture the effect of phrases like 'tasty fish burger', we now add trigrams (i.e. all triples of consecutive words) to the unigrams+bigrams model. The rest of the computations for building the feature matrix are the same as before. Note, however, that the same trigram would rarely occur across different reviews, because two different people are unlikely to use the same 3-word phrase in their reviews. Therefore, the results of this model are not expected to be very different from the unigrams+bigrams model.

The total number of features in this case is 31,677,669.

4.2.4. Latent Semantic Indexing (LSI)

Latent Semantic Indexing (LSI) (Hofmann, 1999) is a more sophisticated method of lexical matching, which goes beyond exact matching of words. It finds 'topics' in reviews, which are words having similar meanings or words occuring in a similar context. In LSI, we first construct a word-review matrix M, of size $m \times t$, using the unigrams model, and then do Singular Value Decomposition (SVD) of M.

$$SVD(M) = U \cdot S \cdot V^T$$

The SVD function outputs three matrices: the word-topic matrix U of size $m \times m$, the rectangular diagonal matrix S of size $m \times t$ containing t singular values, and the transpose of the topic-review matrix V of size $t \times t$. We use V as the feature matrix.

The singular values matrix S has t non-zero diagonal entries that are the singular values in decreasing order of importance. The columns of S correspond to the topics in the reviews. The i'th singular value is a measure of the importance of the i'th topic. By default, t equals the size of vocabulary (i.e. 171,846). However, the first t^* topics can be chosen as the most important ones, and thus the top t^* rows of V can be used as the feature matrix. Determining the value of t^* is crucial, and this can be done by examining a simple plot of the singular values against their importance, and looking for an 'elbow' in the plot.

4.3. Supervised Learning

To train our prediction models, we use four supervised learning algorithms.

4.3.1. Logistic Regression

In logistic regression (Freedman, 2009), the conditional probability function $P(s|\mathbf{r})$ is modelled, where $\mathbf{r}$ is a feature vector for review r and s belongs to the set of class labels $\{1, 2, 3, 4, 5\}$. Then, given a new feature vector $\mathbf{r}^*$ for a new review r^*, this probability function is computed for all values of s, and the s value corresponding to the highest probability is output as the final class label (star rating) for this review.

4.3.2. Naïve Bayes Classification

A Naïve Bayes (Ng and Jordan, 2002) classifier makes the Naïve Bayes assumption (i.e. it assumes conditional independence between any pair of features given some class) to model the joint probability $P(\mathbf{r}, s)$ for any feature vector $\mathbf{r}$ and star rating s. Then, given a new feature vector $\mathbf{r}^*$ for a new review r^*, the joint probability function is computed for all values of s, and the s value corresponding to the highest probability is output as the final class label for review r^*.

In this paper, we use multinomial Naïve Bayes classification, which assumes that $P(\mathbf{r}_i|s)$ is a multinomial distribution for all i. This is a typical choice for document classification, because it works well for data that can be turned into counts, for example weighted word frequencies in the text.

4.3.3. Perceptrons

A perceptron (Rosenblatt, 1957) is a linear classifier that outputs class labels instead of probabilities. It uses a gradient-descent-like rule to iterate over the training set multiple times, in order to re-classify any misclassified examples, until all of them have been classified correctly. For linearly separable data, a Perceptron Convergence Rule states that a solution will always be found after some finite number of iterations. For data that is not linearly separable, there will be oscillation, which can be detected automatically.

Perceptron solutions may be non-unique, because the margin of the linear decision boundary is ignored. In our experiments, we set the number of iterations to be 50 (a typical choice), that is, the classifier loops over the entire training set 50 times.

4.3.4. Linear Support Vector Classification (SVC)

Support Vector Machines (SVM) (Tsochantaridis et al., 2004) are enhanced versions of perceptrons, in that they eliminate the non-uniqueness of solutions by optimizing the margin around the decision boundary, and handle non-separable data by allowing misclassifications. A parameter C controls overfitting. When C is small, the algorithm focuses on maximizing the margin, even if this means more misclassifications, and for lage values of C, the margin is decreased if this helps to classify more examples correctly.

In our experiments, we use linear SVMs for multiclass classification. The tolerance of the convergence criterion is set to 0.001. For each feature extraction method, we do internal 3-fold cross validation to choose the value of C that gives the highest accuracy. It turns out that $C = 1.0$ works best every time.

4.4. Performance Metrics & Implementation Details

We use 80% of the dataset for training, and 20% for testing. For each of the sixteen prediction systems, we perform 3-fold cross validation on the training set and compute two metrics, Root Mean Squared Error (RMSE) and accuracy, for the training fold as well as the validation fold.

All the implementation is done on an Intel Core i5 CPU with 4 cores (1.6 GHz each), 8 GB RAM and 64 bit Ubuntu 12.04 operating system. The programming language used is Python, and extensive use is made of its libraries numpy, scipy and scikit-learn.

5. Results and Analysis

In this section, we present the results for the four feature extraction methods separately. For each method, we show an RMSE graph and an Accuracy graph; each graph contains plots for the four classifiers. We then analyze these results to choose the best of the sixteen systems, and finally we evaluate the chosen system on the test set.

5.1. Unigrams

Figures 3(a) and 4(a) show the performance of the four classifiers on unigrams' features. The total number of available features is 171,846 and we do not know how many of these are useful, so we plot the RMSE and the accuracy against the top[1] x number of features. We see that as the number of features increases, the training-fold RMSE for each classifier increases and the training-fold accuracy of each classifier decreases, but the validation-fold RMSE and the validation-fold accuracy level off at about 10,000 features. The only exception is the Naïve Bayes classifier, whose RMSE reaches a minimum value around 10,000 features, but then starts rising again.

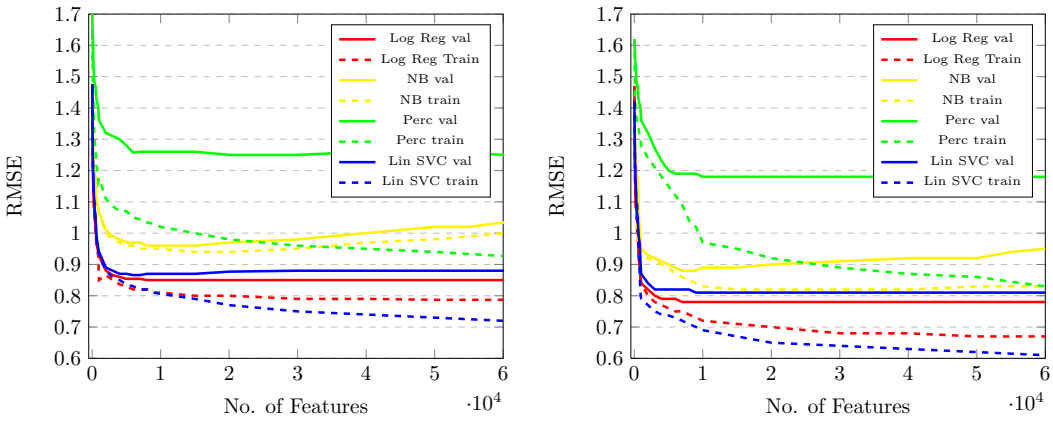
Clearly, perceptrons perform the worst, achieving a lowest RMSE of 1.25 and the highest accuracy of 43%. The Naïve Bayes classifier is the second worst, with the best RMSE and accuracy values of 0.96 and 52%. The performances of Linear SVC and logistic regression are not significantly different on the validation fold. Linear SVC's best RMSE and accuracy scores are 0.87 and 57% , while those for logistic regression are 0.85 and 58%. So, logistic regression has the best performance for unigrams.

Note that a random (coin-flip) classifier, that assumes no knowledge of the data distribution, would have an accuracy of 20%, because we have 5 classes. Logistic regression improves the random classifier accuracy by almost 300%.

5.2. Unigrams & Bigrams

Figures 3(b) and 4(b) show the performance of the four classifiers on the top x number of features obtained from unigrams & bigrams. The total number of available features in this case is over 7 million. We see that the RMSE and accuracy improves for every classifier, compared to Figures 3(a) and 4(a). This is because bigrams occur frequently enough in the corpus, capture a lot of information that unigrams cannot, and give

[1] For example, top 10 features would be the features with the top 10 TF-IDF weights.



(a) Unigrams (b) Unigrams & Bigrams

Figure 3. RMSE plots for (a) Unigrams, and (b) Unigrams & Bigrams

much more meaningful features. However, the overall trends are quite the same for the training and validation folds. Perceptrons are still the worst, followed by the Naïve Bayes classifier. Linear SVC and logistic regression are quite close again. Linear SVC's best RMSE and accuracy scores are 0.81 and 63%, while those for logistic regression are 0.78 and 64%. Logistic regression wins again.

5.3. Unigrams, Bigrams & Trigrams

The results for this feature extraction method are almost exactly the same as those for the 'Unigrams & Bigrams' method (Figures 3(b) and 4(b)), and we omit the graphs due to brevity of space. The best RMSE and accuracy scores are 0.78 and 64%, achieved by logistic regression.

Adding trigrams to the previous model does not help, because trigrams repeat rarely. It is unlikely that two different user would use the exact same 3-tuple to describe a restaurant. The TF-IDF weighting technique weights almost all the 3-tuples as very rare, therefore they are not very useful as features.

5.4. Latent Semantic Indexing (LSI)

Figure 5 shows the results of the experiments with LSI. Figure 5(a) is a plot of the 1000 highest singular values. We see that the graph starts leveling out at 200. This means that the top 200 topics in the reviews are the most important ones for training the model. Next, we evaluate each classifer on feature vectors of length up to 200; the performance of each classifer is shown in Figure 5(b) and (c), for the validation fold[2].

Figures 5(b) and (c) show some interesting patterns. Perceptrons perform the worst as usual, however we see two spikes in the performance around 170 and 200 features, where the RMSE and accuracy suddenly improve. It is not clear why this happens, but it suggests that we should consider more than 200 features to see if more of these spikes occur and improve the best scores for perceptrons. The plots for logistic regression show another interesting pattern. As the number of features increases, the RMSE remains constant around 1.3, but the accuracy increases from 0.34 to 0.43. One explanation for this apparent anomaly is that, more examples get classified accurately, but the RMSE for the misclassified examples increases and overshadows the decrease in the RMSE due to better accuracy.

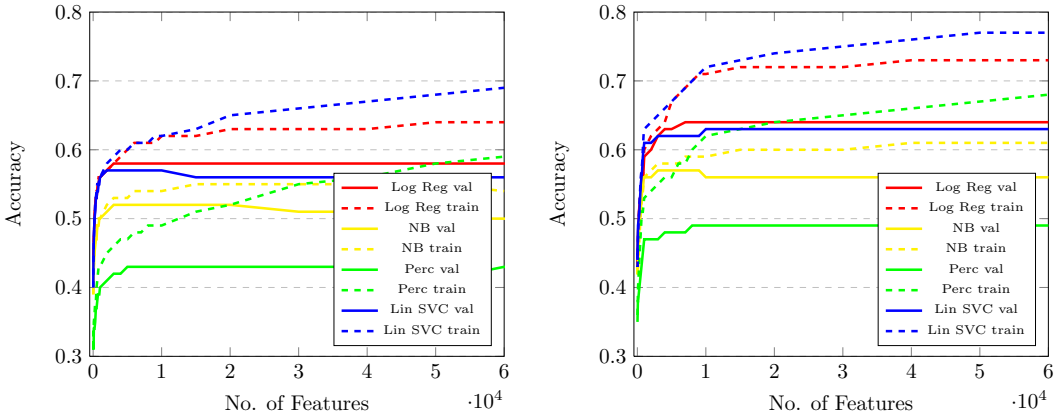
A similar pattern is seen for Linear SVC, and could be explained as above. The upward trend in its accuracy suggests we should consider more than 200 features. Due to time constraints, we leave this experiment as future work.

5.5. The Best Model: Performance on the Test Set

Based on the results in Figure 3, 4 and 5, we can see that Logistic Regression achieved the highest accuracy of 64% using the top 10,000 Unigrams & Bigrams as features, followed very closely by Linear SVC which achieved 63% accuracy using the top 10,000 Unigrams & Bigrams.

Next, we evaluate these two systems on the test set.

[2]We could not obtain the training-fold RMSE and accuracy results due to time constraints.



(a) Unigrams (b) Unigrams & Bigrams

Figure 4. Accuracy plots for (a) Unigrams, and (b) Unigrams & Bigrams

For Linear SVC, the RMSE and accuracy scores are 1.05 and 56%, and for logistic regression, the scores are 0.92 and 54%. These test-set scores are slightly worse than the validation-fold scores, and possibly indicate overfitting; to fix that, we would need to add/adjust the regularization parameters and re-run all the experiments. This is left as future work.

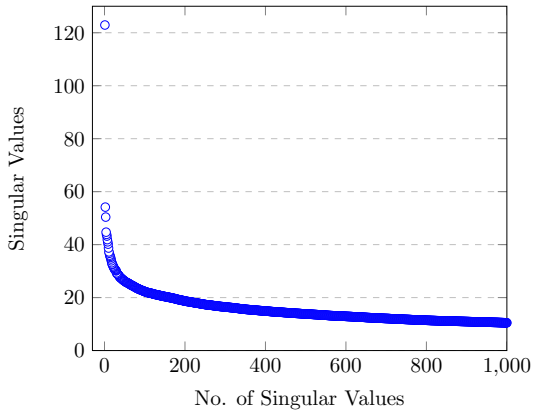
6. Conclusions & Future Work

In this paper, we tackle the Review Rating Prediction problem for restaurant reviews on Yelp. We treat it as a 5-class classification problem, and examine various feature extraction and supervised learning methods to construct sixteen prediction systems. Experimentation and performance evaluation through k-fold cross validation yields one system, Logistic Regression on the set of top 10,000 features obtained from Unigrams & Bigrams, that exhibits better predictive powers than the others. Our system can be used to generate star ratings on review websites where users can write free-form text reviews without giving a star rating.

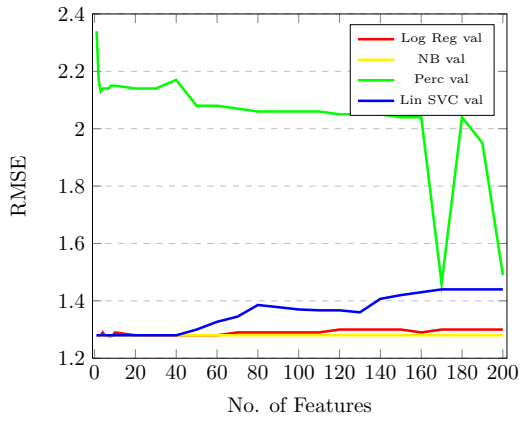
Though the methods tested in this paper are extensive, they are by no means exhaustive. In fact, there are many avenues for improvements and future work. We list them below.

1. We can try more sophisticated feature engineering methods, such as Parts-of-Speech (POS) tagging and spell-checkers, to obtain more useful n-grams. For example, instead of considering all possible bigrams, we can extract all the adjective-noun pairs or all the noun-noun pairs to get more meaningful 2-tuples. This would significantly help with the system's efficiency as well. Our current implementation in Python is quite slow (each plot in Figures 3, 4 and 5 took 36 to 48 hours) because we deal with up to 32 million features. It is also memory-intensive. Choosing a smaller number of features more carefully will help tremendously with this bottleneck.
2. We can try more elaborate experiments for LSI that consider more than 200 features. Moreover, instead of performing singular value decomposition of unigrams only, we can add other n-grams. Another possibility is to perform SVD on specific text-constructs only, such as the nouns or the adjective-noun pairs. It would be interesting to see how the results change.
3. A possibly better alternative for logistic regression that should be tried is ordered/ordinal logistic regression [3]. It is an extension of logistic regression that specifically caters to classification problems where the class labels are ordered. So, in our case, this model takes into consideration the fact that the class labels 1 and 2 are closer to each other, than the labels 1 and 4.
4. All the prediction models that we use in this paper are linear. That is, the hypothesis function is a linear function of the parameters. It would be interesting to experiment with non-linear models, e.g. polynomial regression, SVC with a non-linear kernel, etc. Alternatively, to get a non-linear decision boundary, we could find a linear decision boundary in an expanded feature space; for example, the feature vectors $\mathbf{r}_i$ could be replaced with $\phi(\mathbf{r}_i)$, where ϕ is called a *feature mapping*.
5. As mentioned in Section 5.5, our test-set performance was worse than the validation-fold per-

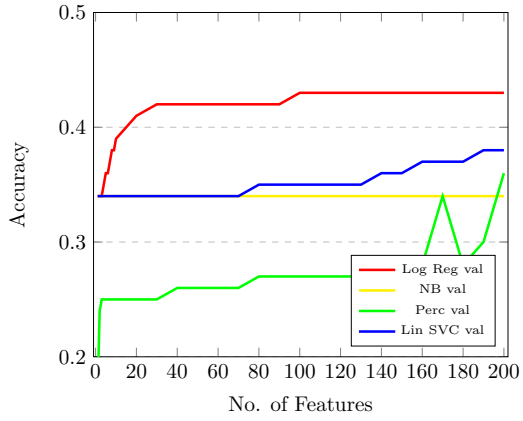
[3] http://www.norusis.com/pdf/ASPC_v13.pdf.



(a)



(b)



(c)

Figure 5. Latent Semantic Indexing (LSI)

formance, and this could be due to over-fitting. To fix this, we can introduce/adjust the regularization parameters in our models using internal cross-validation to get improved performance.

6. For feature extraction, we could try topic modelling techniques such as Latent Dirichlet Allocation (Blei et al., 2003), Non-negative Matrix Factorization (Tandon and Sra, 2010) and/or Independent Component Analysis (Stone, 2004). We could also use one or more of the sentiment analysis techniques mentioned in Section 2, and possibly combine them with the semantic analysis techniques we use in this paper to improve the results.
7. To get a more detailed performance evaluation, we can try other metrics, such as precision, recall, F-score and confusion matrix. Also, for probabilistic models, we can analyse the confidence scores.
8. We can compare our classifiers' preformance with the performance of the traditional recommendation techniques such as collaborative filtering.
9. We chose to do 3-fold cross validation in our experiments because 10-fold cross validation turned out to be very expensive in terms of time and memory. It would be worthwhile to try 10-fold cross validation and see if it yields different and/or better results. To deal with the time and RAM bottleneck, we could try parallel processing over clusters, such as those provided by Sharcnet[4].
10. Another avenue for future work is to test how our prediction models would perform on other business categories, such as shopping, travel, etc.

[4] https://www.sharcnet.ca.

References

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

P. Y. Chen, S. Y. Wu, and J. Yoon. The impact of online recommendations and consumer feedback on sales. In *Proceedings of the International Conference on Information Systems*, pages 711–724, 2003.

Yelp Dataset, 2014. URL http://www.yelp.com/dataset_challenge.

C. Dellarocas. The digitization of word of mouth: promise and challenges of online feedback mechanisms. *Management Science*, 49(10):1407–1424, 2003.

Mingming Fan and Maryam Khademi. Predicting a business star in yelp from its reviews text alone. *arXiv preprint arXiv:1401.0864*, 2014.

David Freedman. *Statistical models: theory and practice*. Cambridge University Press, 2009.

Gayatree Ganu, Noemie Elhadad, and Amélie Marian. Beyond the stars: Improving rating predictions using review text content. In *WebDB*, 2009.

T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999)*, pages 50–57, Berkeley, CA, 1999.

Cane WK Leung, Stephen CF Chan, and Fu-lai Chung. Integrating collaborative filtering and sentiment analysis: A rating inference approach. *Proceedings of the ECAI 2006 workshop on recommender systems*, pages 62–66, 2006.

F. Li, N. Liu, H. Jin, K. Zhao, Q. Yang, and X. Zhu. Incorporating reviewer and product information for review rating prediction. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 1820–1825, 2011.

A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naïve bayes. In *Advances in Neural Information Processing Systems*, pages 842–848, 2002.

L. Qu, G. Ifrim, and G. Weikum. The bag-of-opinions method for review rating prediction from sparse text patterns. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 913–921, 2010.

Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

James V Stone. *Independent component analysis*. Wiley Online Library, 2004.

Rashish Tandon and Suvrit Sra. Sparse nonnegative matrix approximation: new formulations and algorithms. *Max Planck Institute for Biological Cybernetics*, 2010.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning (ACM 2004)*, pages 104–111, 2004.