Stuti Polra Rajdeep Singh Lather

Predicting User Ratings for Businesses on Yelp

I. Introduction

Task Description

The task for this project consists of using Yelp reviews and information about Yelp users to predict the star rating a user would give to a business. Yelp is a website that allows users to post text reviews about various businesses they visit and also leave star ratings that reflect their experiences. Using machine learning to predict user ratings can prove helpful for the businesses in assessing their services and better understanding their customers. It can help them fix the things their customers are displeased with or capitalize on the things that their customers are satisfied with. In this project we perform sentiment analysis on the text reviews left by the users to obtain a sentiment feature for the text, and use that in addition to other review metadata and information about the user to predict the star rating associated with each review. We use the spark-nlp library for sentiment analysis and the multilayer perceptron from pyspark's machine learning library to obtain the star rating prediction.

Background Research

Our research on performing sentiment analysis on large datasets lead us to the work of Akshi Kumar, et. al.[1] where the authors use corpus based and dictionary based methods to determine sentiments of opinion words in tweets. Next, we tried to determine the best way to do sentiment analysis on Spark and found the Spark NLP[3] library for doing Natural Language Processing in Spark. In order to determine if this library was adequate for our task we examine a review by Saif Addin Ellafi[2] that benchmarked the performance and scalability of Spark NLP. Saif's review notes that Spark NLP offers excellent scaling and good performance for NLP tasks. Next, we looked at a way to get sentence embeddings for our NLP pipeline. Since, most reviews have multiple sentences, we needed an embedder that created a single vector for the entire review text. For this purpose we used the Universal Sentence Encoder[4]. Finally, we looked at the viability of using auxiliary data alongside the sentiment of a review text to do rating prediction. The work of Duyu Tang, et. al. [5] titled 'User modeling with neural network for review rating prediction' proved helpful as it employs user information to interpret a review and predict the user's rating.

Dataset Description

The Yelp dataset (9.74GB) has been used for this project. It consists of 8,021,122 million reviews left by users for 209,393 businesses across 10 metropolitan areas. The reviews file contains the user's id, the business' id, the text review, the star rating, when it was posted, and the number of userful, funny and cool votes the review received from other Yelp users. The users file contains the user's name, id, how long they have been a Yelp member, their review count, friends, the average stars they give, various compliments they received, the number of fans they have and the number of useful, cool or funny votes they have gotten from other users. Although the dataset provides information on the businesses, tips the users have left, and pictures the users posted, we have only used information about the review and the user (8.93GB) to come up with star rating predictions. The dataset is publicly available on Kaggle at: https://www.kaggle.com/yelp-dataset/yelp-dataset

II. Methodology

The two main stages in the methodology employed for this project are feature engineering and model selection on the multi-layer perceptron model used to obtain the rating predictions.

Feature Engineering

In the feature engineering stage, we decide which features are useful for predicting the star ratings since providing features unrelated to the rating can yield poor performance. From the available user information, we chose to use the number of reviews the user has left, the average stars the user gives which can explain the user's rating behavior (harsh or lenient), the number of fans the user has and the number of useful votes the user has gotten from other users. From the available information on a review, we choose to use the review text, whose sentiment would correlate with the star rating, and the number of useful, funny and cool votes the review received from other users as these can establish the credibility of the review. Since using the entire review text as a feature is infeasible, we use the text to obtain the sentiment and use this sentiment (positive or negative) as a feature.

Sentiment Analysis from Review Text

To obtain the sentiment of each text review, we use the spark-nlp library [3] which provides pre-trained models and pipelines for performing natural language processing with neural networks. We created a custom pipeline with three stages: document assembler, sentence encoder, and sentiment analyzer. The DocumentAssembler takes the raw text reviews and creates a Document object for each review with the necessary annotations (begin, end, result, sentence embeddings) that are either filled in at this stage, or will be filled in in the next two stages. In the next stage, we use the pre-trained UniversalSentenceEncoder neural network model [4] to obtain sentence embeddings for each of the text reviews. We use a sentence encoder as opposed to a word encoder to obtain a single vector for the entire text review. This model has been trained to work with greater-than-word-length text such as sentences, phrases and short paragraphs. It encodes the variable length text into a 512 dimensional vector. In the final stage, a pre-trained AnalyzeSentimentDL neural network model uses the sentence embeddings to obtain sentiment (positive or negative) for the reviews.

We experimented with two pre-trained sentiment analysis models: one trained on IMDB movie reviews, and one trained on data from twitter. To assess the suitability of these models for our task, we measure the accuracy of the sentiment predictions they generate by thresholding (less than 2.5 stars indicates negative sentiment, more than 2.5 stars indicates positive sentiment) the star ratings to get sentiment labels for each review. The table below shows the accuracy values of the sentiment predictions when using the two different models:

Sentiment Analysis Model	**Accuracy of Sentiment Predictions**
Pre-trained on IMDB data	0.8589
Pre-trained on twitter data	0.7484

Since the language used in short tweets is quite different from the kind of language used in longer reviews, the model trained with twitter data does not perform as well on text reviews. Although the IMDB model is trained on movie reviews and our reviews are for various businesses, they have a similar domain which yields better accuracy. Since the model pre-trained on IMDB data gave us a higher accuracy on the sentiment predictions, we use the sentiment predictions from this model as a feature for our multilayer perceptron model.

Model Selection: Multi-Layer Perceptron

To obtain the star rating predictions, we employ the pyspark machine learning library's implementation of the MultilayerPerceptronClassifier. Since the machine learning library does not offer a neural network implementation for regression, we adapted the classifier to fit our regression task. As the star ratings in the data set are an integer value (ranging from 0 to 5), we can use a multilayer perceptron classifier with 6 classes on the output end corresponding to the star rating predictions of 0, 1, 2, 3, 4 or 5. The multilayer perceptron model, also known as a neural network, has an input layer where the number of units are equal to the number of features in the input; it can have multiple hidden layers with a varying amount of units, and the number of units in its output layer is equal to the number of labels desired. Each layer in the network has a sigmoid activation function, and the output layer has a softmax that gives the probability of the sample belonging to each of the classes, such that the class with the highest probability is selected as the prediction. The evaluation metrics used to assess the performance of the model are root mean square error and mean square error. Although we use the classifier to obtain predictions, regression metrics must be used to evaluate the performance. A classification metric like accuracy would not be suitable since it only counts the number of predictions that exactly match the labels, but we must assess how close our predictions are to the labels--which requires using regression metrics.

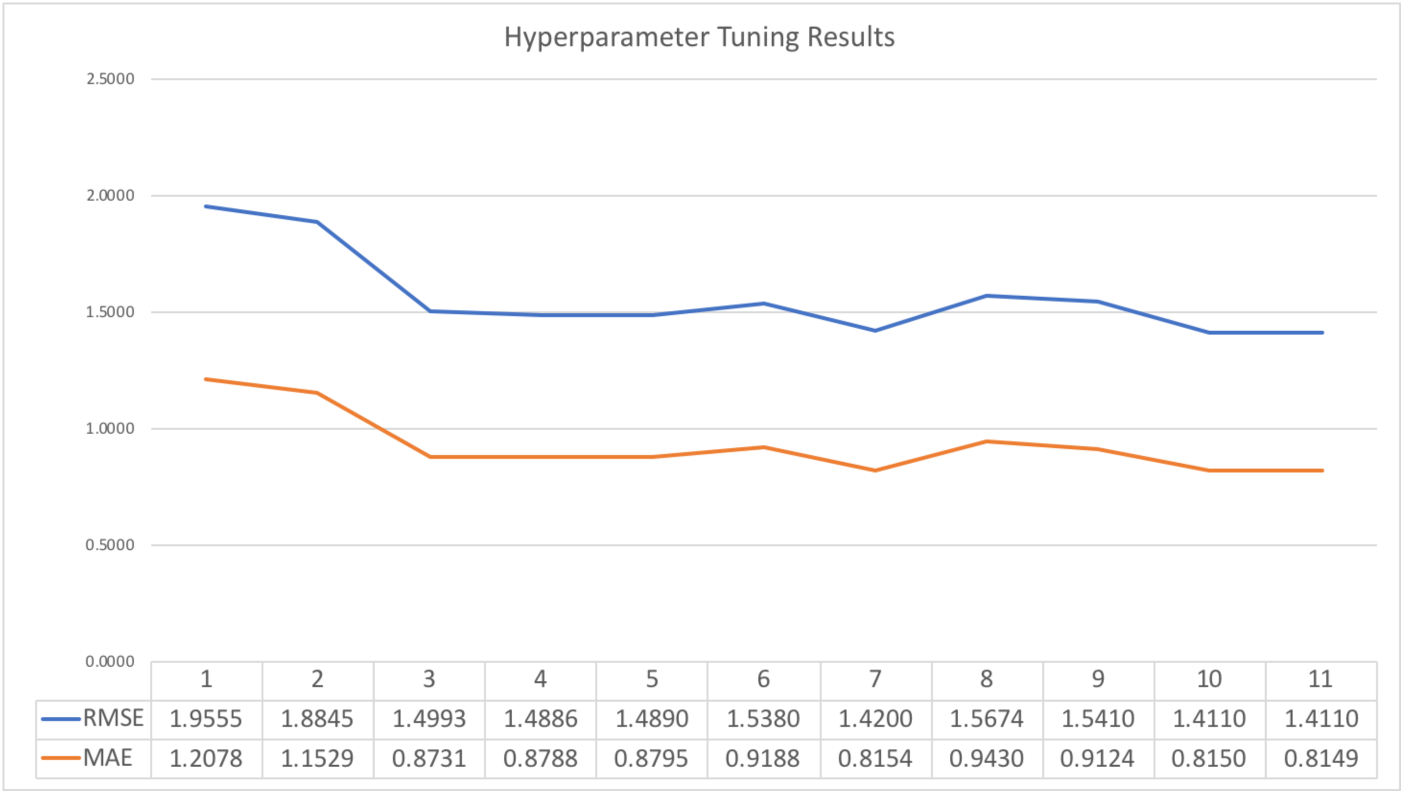
Model selection and parameter tuning allows neural networks to find optimal weights and prevents overfitting. Since neural network training can take a long time, cross-validation is not a feasible approach to tune and validate the model. The most common approach for tuning neural networks is performing several runs with development and validation sets created from the training data. Following this convention, we split our data into train (70%) and test (30%) sets, and then further split the train set into development (70%) and validation (30%) sets. We tune various parameters using the development set to train the model, and the validation set to assess its performance. With the optimal parameters that yield the best performance on the validation set, we train the network on the entire training set, and evaluate the performance on the test set.

The parameters for the multilayer perceptron that we experiment with and tune are: number of hidden layers, the sizes of those hidden layers, the solver or algorithm (l-bfgs and gradient descent) used to find the optimal weights of the network, the step size used for each iteration of optimization, and the maximum of number of iterations to train for.

Hyperparameter Tuning Results

We observe that with the l-bfgs optimization algorithm, the network is able to converge to the optimal weights sooner (in less iterations) than when using gradient descent. The performance degrades when using more layers as the network is becoming more complex and starting to overfit the data. Since the network with a single hidden layer performs better, we experiment with the number of units in the hidden layer, and find that having more units yields better performance. Considering the step size used in the optimization algorithm, we found that 0.2 is too large and the network misses the optimum, 0.01 is too small and the network does not reach its optimum, and with 0.1 it is able to reach the optimum and yield the best results. The parameters indicated in bold in the table below gave the best performance on the validation set while tuning the model. The graph presented below shows the performance of the network for each of the 11 parameter settings noted in the table below.

S. No.	maxIter	layers	stepSize	solver	RMSE	MAE
1	80	[8, 6, 6]	0.1	gd	1.9555	1.2078
2	200	[8, 6, 6]	0.1	gd	1.8845	1.1529
3	80	[8, 6, 6]	0.1	l-bfgs	1.4993	0.8731
4	40	[8, 6, 6]	0.1	l-bfgs	1.4886	0.8788
5	40	[8, 6, 6]	0.01	l-bfgs	1.4890	0.8795
6	40	[8, 6, 6]	0.2	l-bfgs	1.5380	0.9188
7	40	[8, 18, 6]	0.1	l-bfgs	1.4200	0.8154
8	80	[8, 6, 12, 6]	0.1	l-bfgs	1.5674	0.9430
9	80	[8, 6, 4, 6]	0.1	l-bfgs	1.5410	0.9124
10	80	[8, 12, 6]	0.1	l-bfgs	1.4110	0.8150
11	**40**	**[8, 12, 6]**	**0.1**	**l-bfgs**	**1.4110**	**0.8149**



	1	2	3	4	5	6	7	8	9	10	11
RMSE	1.9555	1.8845	1.4993	1.4886	1.4890	1.5380	1.4200	1.5674	1.5410	1.4110	1.4110
MAE	1.2078	1.1529	0.8731	0.8788	0.8795	0.9188	0.8154	0.9430	0.9124	0.8150	0.8149

III. Results and Discussion

The table below indicates the root mean square error and the mean absolute error the network obtained on the test set after training on the train set with optimal parameters.

Final Testing Results

Optimal Model Parameters	**RMSE**	**MAE**
maxIter=40, layers=[8, 12, 6], stepSize=0.1, solver='l-bfgs'	1.3927	0.8208

Comparing the results of the model on the full train set we observe that the values for RMSE and MAE are around the values obtained during validation when the development and validation sets were used. The values are expected to be slightly larger as the model is being trained on more data. Both RMSE and MAE provide the model's average prediction error and do not consider the direction of the error, so predicting a star above or a star below the actual star rating label would be penalized equally. However, since RMSE squares the errors before averaging them, it gives a higher weight to larger errors. For example, if a prediction was two stars away from the label, it would have a higher weight than a prediction that was one star away from the label. Keeping this in mind, we interpret our results as the following: if equal weight is given to all errors, our model predicts within 0.82 stars of the label, and if more weight is given to larger errors, our model predicts within 1.39 stars of the label. The results are reasonable as our model predicts ratings within a star rating of the actual label.

Text Review	**Predicted Sentiment**	**Predicted Rating**	**Actual Rating**
Solid place to go for dinner! Great for sharing. I really liked the fried brie, it was my favourite of the evening. Really fun environment to have dinner with friends!	Positive	4	4
Rude service and gross coffee, as for the bagels... I'm a New Yorker, so these bagel-shaped bread things being called "bagels" are laughable to me. Also, why are they charging an arm and a leg for basic bagels + eggs, things that are supposed to be cheap?? No, thanks.	Negative	1	2

The above results show a sample output of our model, a larger sample with all the relevant columns can be found in the output.txt file. We observe the sentiment predicted by our system in the first stage correlates with the predicted and actual star ratings.

IV. Conclusion

In this project we obtain star rating predictions of Yelp reviews left by users for businesses. We employ pre-trained natural language processing models to obtain sentiment for the review text, which alongside other review and user features are used by pyspark's multilayer perceptron model to predict star ratings. Some directions for future work include transfer learning on the NLP portion to fine tune the pre-trained sentiment analysis model with the review text, and use of a more robust deep learning framework in conjunction with spark. Such a framework would allow for experimentation with early stopping, dropout and attention mechanisms in the neural network training.

V. References

[1] Sentiment Analysis on Twitter - By Kumar, Akshi & Sebastian, Teeja. (2012). International Journal of Computer Science Issues. 9. 372-378.

[2] Comparing production-grade NLP libraries: Accuracy, performance, and scalability - By Saif Addin Ellafi.
oreilly.com/content/comparing-production-grade-nlp-libraries-accuracy-performance-and-scalability/

[3] Spark NLP: State of the Art Natural Language Processing, nlp.johnsnowlabs.com

[4] Universal Sentence Encoder - By Cer, D.M., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y., Strope, B., & Kurzweil, R. (2018). ArXiv, abs/1803.11175.

[5] User modeling with neural network for review rating prediction - By Duyu Tang, Bing Qin, Ting Liu, and Yuekui Yang. (2015). In Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15). AAAI Press, 1340–1346.