Yelp Review Rating Prediction

Avishek Ghosh† (UID: 205333248), Madison Kohls† (UID: 205302850),
Rebecca Xu† (UID: 405604229), and Brandon Zhao† (UID: 005335899)
†Department of Statistics UCLA

Abstract

Yelp is an online business review platform that allows users to leave ratings and reviews. We will be analyzing the relationship between the rating given and review written with the ultimate goal of review rating prediction, i.e. predicting the star-rating from the review's sentiment using models such as logistic regression, KNN, LDA, QDA, and random forests.

⋆⋆⋆⋆⋆

1 Introduction

Looking for a new place to eat tonight? Yelp is a popular online platform for rating and reviewing businesses. With over 200 million reviews on the app, Yelp has become both a helpful and reliable guide for consumers to choose restaurants, services, and so on. Generally, a Yelp review will include the following two components:

1. 5 star rating: An overall rating (on an integer scale of 1-5) with 5 being the best and 1 being the worst.

2. Review: Some sentences detailing the experience of the business.

In our project, we are tasked with analyzing a Yelp dataset [1] with over 53,000 reviews and ratings. We aim to explore the relationship between ratings and reviews and ultimately formulate statistical models to predict the star rating through the review's words and sentiment.

2 Prepreprocessing Step

To achieve our task of predicting Yelp Review ratings from Yelp data, we started with two json files containing business information and reviews from Yelp.

We preprocessed our data by first using a basic Python script to load and append both the business and review data from their respective json files, and then merged them both together. To shrink the amount of data we have, we then subsetted the dataset to only reviews in California and from those who have reviewed over 150 businesses before. From this we ended up with a dataset on the user-business grain, containing information about the user, the business, the rating (number of stars from 1 to 5), and the review. We then created a binary column 'is highly rated' that is True if the review was a 4 or 5, and False otherwise.

2.1 Descriptive Statistics

To get a better understanding of the data we have, we then constructed some descriptive statistics.

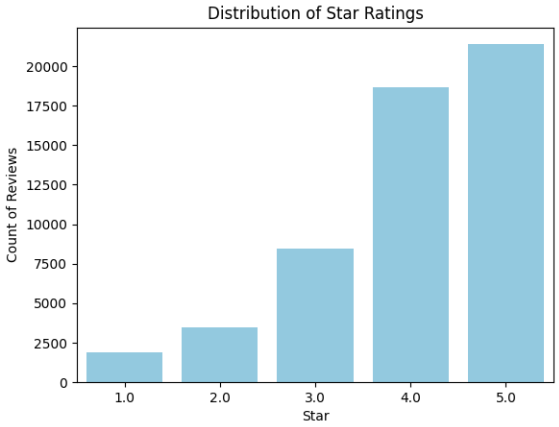

Figure 1: Distribution of Star Ratings

We can see that a majority of our ratings are 4's and 5's, which may pose an issue as it means our dataset is imbalanced.

Table 1: Top Cities with the Most Reviews

City	Count
Santa Barbara, CA	42532
Goleta, CA	6009
Carpinteria, CA	2557
Isla Vista, CA	1425
Montecito, CA	848

We can see that most of the reviews are concentrated around the Santa Barbara region of California. Knowing this, we may not be able to extend the results of our model to reviews in other regions of California since some bias may be present in the dataset.

Figure 2: Frequency of Words in Reviews

We can see that, excluding simple stop words like "the" or "a", "place" and "food" appear the most in all the written reviews. However, these words don't seem as useful as common words like "good" and "delicious" to attribute to a certain rating.

2.2 Clustering of Users

To get a better understanding of the users within the dataset, we decided to run a K-Means model on the number of reviews, average stars, average review length, and other variables for each user. K-means [9] is an unsupervised learning model that clusters data points together. Doing this process will help us better understand the different types of reviewers in the dataset.

In K-means, the number of clusters, K, must be predetermined. In order to determine the optimal number of clusters to create, we use the elbow method. We create a model for each possible K from 1 to 15. Then, we measure the inertia for each K. In-

ertia measures the collective distance between each data point and its centroid. While inertia will always decrease when we increase K, we can determine the optimal K by seeing when inertia starts declining more linearly. This will correspond with the "elbow" of the graph, hence the name of this method.

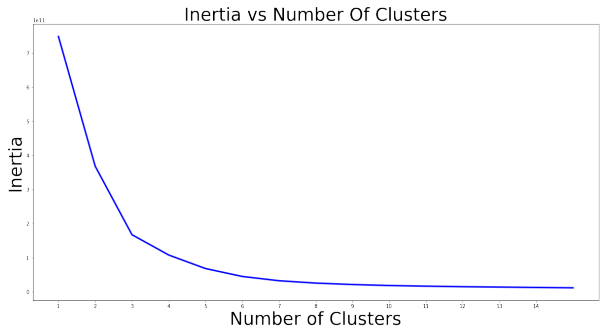

Figure 3: Inertia for every K

Using this method, we can see that inertia starts declining more linearly after increasing K beyond three. Thus, the ideal number of clusters is three. We then generate a final K-means model using K = 3. The below table gives information on select variables for each of the centroids/clusters.

Table 2: Centroids of Clusters

	# Reviews	Stars	Review Length	# Users
1	391.35	3.87	680.69	10876
2	4110.63	3.86	1778.78	8
3	1743.75	3.93	1075.27	170

From the results of our clustering model, we can see three types of user behavior. The largest cluster of users have a smaller average number of total reviews and lower review lengths. The smallest cluster of eight users has the largest average number of total reviews and the longest review lengths. Finally, there is an in-between cluster containing slightly more users with in-between values for average number of reviews and review length. An important point to note is that these clusters all have very similar average stars given.

From these categories, we can see that there are many casual users who just occasionally leave reviews, and a small group of dedicated individuals who leave many, long reviews. Furthermore, this gives us more insight into the dataset that we are working with. It may be the case that a large proportion of reviews in this dataset were written by just a small number of very active users. Because every individual has their own writing styles and word preferences, this imbalance in the dataset may have an effect on future analyses and models we run.

2.3 Feature Engineering: Sentiment Lexicon and PCA

We cannot work with the review data in its raw form, so we performed some feature engineering in order to extract any useful information from the review. We removed all stop words and first used TF-IDF to show words between a minimum (0.1) and maximum (0.7) document frequency (how often a document contains a certain word). However, we ended up with a vector of only 73 words, where many of the words seemed irrelevant to a particular rating (words such as 'also', 'price', and 'menu'). To combat this, we constructed a sentiment lexicon by using a pre-made text document with negative and positive words to vectorize all positive and negative words in each review and converted the array into a dataframe. We ended with 4024 columns of words.

Because 1) our task is a prediction task, 2) our word columns may be highly correlated, and 3) we have very high dimensionality in

our dataset from the word columns, we then performed PCA (Principal Component Analysis) on our word columns to reduce the dimensionality of our dataset. PCA [7] finds linear combinations of the given raw predictors that are orthogonal to each other and contain the most variance in order to reduce the overall number of predictors while still containing as much variation as possible. In order to determine the best number of predictors, we calculated all 4024 components in order by variance explained and then plotted their cumulative variance explained by each component.

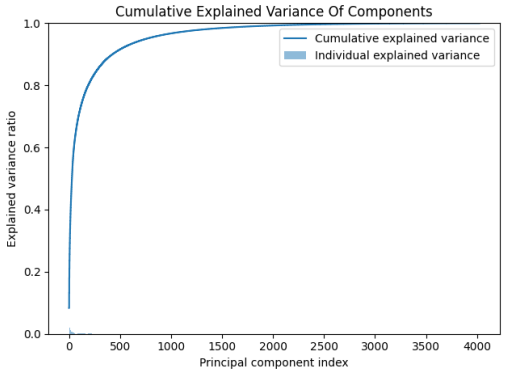

Figure 4: Principal Components by Proportion of Variance

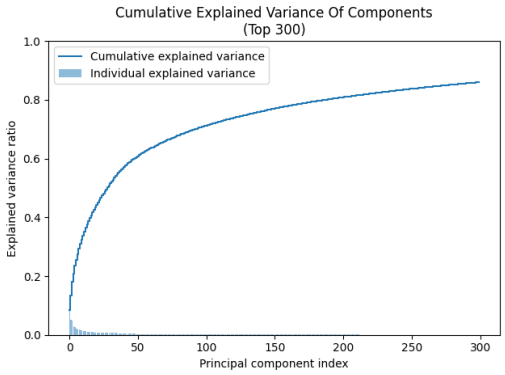

Figure 5: Top 300 Principal Components by Proportion of Variance

We can see in Figure 4 that there seems to be a plateau in variance explained after 1500 components. However, that would not be a sizable reduction in dimensionality. In Figure 5 and from outputting the cumulative variance, around 80% of the variance can be explained by the first 200 components. For this reason, we selected the top 200 components.

2.4 Balancing Dataset: SMOTE

Finally, to combat the issue of imbalanced data for training, we wanted to balance the classes in our training set. There are multiple ways of doing this such as under-sampling from the majority class (ratings of 4 or 5 stars). However, the issue with this approach is that we lose a lot of information and the number of training example becomes much smaller than the entire data available. To combat this approach, we used the Synthetic Minority Oversampling Technique, also known as SMOTE [3, 8], to balance the training set. We first performed a 75-25 split on the whole dataset. We chose to not balance the test set because we want to evaluate the model so that it is generalized for the real world, which will have more high ratings (4 or 5 stars) than low ratings (1,2 or 3 stars).

Additionally, we found that SMOTE performed better than a random oversampling method because it generates synthetic samples for the minority class. When we performed the train test split, the train set had 40,383 training examples and test set had 13,462 examples. Of the 40,383 training examples, 29,989 were for high ratings (4 or 5 stars). After we applied SMOTE, the dataset was balanced and there were 29,989 training examples for both high and low ratings. Using SMOTE created 29,989 - 10,394 = 19,595 synthetic data points for the low ratings class.

3 Experiment

To consider the best model for prediction, we used four supervised learning algorithms.

3.1 Logistic Regression

Logistic Regression [4] is a technique that is used to estimate the conditional probability of a binary event occurring given some data. This conditional probability estimation is an extension of the popular Linear Regression which is used for predicting continuous variable.

Often, predictive models fit very closely to the training data. This is called overfitting when the model performs poorly on unseen data. One of the main reasons for this are extreme coefficients. Regularization is a method used to limit large coefficients to generalize a model and prevent overfitting. L1 and L2 Regularization are the two most common regularization techniques. For our logistic regression model, we decided to proceed with L2 regularization because L1 creates sparsity. Since we already performed PCA for reducing the number of predictors, we did not want further variable reduction.

For our model, there was only one hyperparameter that needed tuning. This is the value of C, which controls the weight that model places on the regularization penalty. Higher values of C causes the model to fit more closely to the training data whereas the low values of C gives more weight to the penalty term when coefficients are computed. We tuned the hyperparameter using Grid Search Cross Validation and obtained C=1 to produce the best estimator.

After training the Logistic Regression model, we received the following results from the test set:

Table 3: Logistic Regression Test Results

	Precision	Recall	F1-Score	Support
False	0.5335	0.7667	0.6292	3408
True	0.9072	0.7727	0.8346	10054
Accuracy	–	–	0.7712	13462
Macro Avg	0.7203	0.7697	0.7319	13462
Weighted Avg	0.8126	0.7712	0.7826	13462

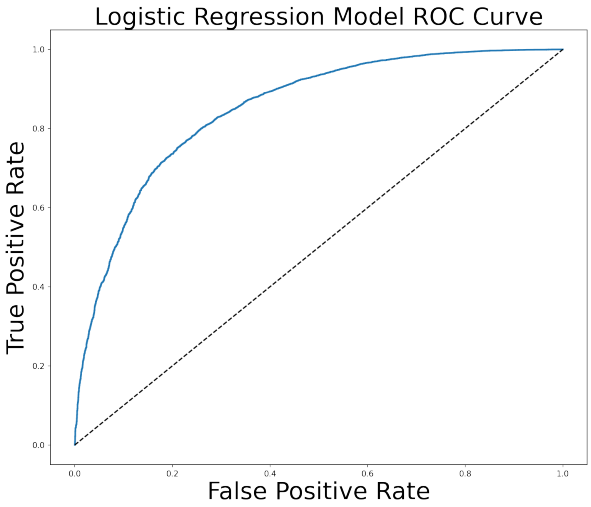

Figure 6: ROC Curve of the Logistic Regression Model

As we can see, the model performs better in classifying into the majority class (True / Reviews of 4 or 5 stars). This is in spite of balancing the training dataset using SMOTE.

3.2 K-Nearest Neighbors

K nearest neighbors [10], or KNN for short, is a supervised learning method that classifies based on the data points closest in proximity using the distance between points. Since KNN utilizes a non-parametric approach, it is a highly flexible model and no assumptions need to be made about our data. This is an advantage for us because we are working with a very large dataset.

In our case, we are classifying points in a high 200-dimension space (since we have

200 principal component predictors). Then within the"K" closest-in-distance neighbors, the data point can be classified based on the its neighbors' majority classification: either "true" for being highly rated (4 or 5) or "false" (3 and below).

After training the model on the SMOTE dataset using five-fold cross validation, we are able to select the optimal "K" value:

Table 4: KNN CV Results

k	Accuracy	Kappa*
5	0.7431724	0.4863450
7	0.7296341	0.4592682
9	0.7210976	0.4421952

*inter-rater reliability statistic

Looking at the accuracy values of each K, K=5 has the highest accuracy of approximately 74% of points correctly classified. Looking at the rightmost column, Kappa is a similar metric to accuracy [2], but also takes into account accuracy as a result of chance or randomness of the dataset. As a general rule, a Kappa value between 0.40 and 0.75 is considered acceptable with a higher value being better. K=5 also has the highest Kappa value of about 0.486. Thus, based on both metrics, K=5 is the best choice for our KNN model with an expected accuracy of almost 75%. Running KNN with K = 5 on our testing dataset, we arrive at the following results:

Table 5: KNN Test Results

	Precision	Recall	F1-Score	Support
False	0.7344	0.3857	0.5058	3408
True	0.6034	0.8702	0.7128	10054
Accuracy	–	–	0.6366	13462
Macro Avg	0.6689	0.6279	0.6092	13462
Weighted Avg	0.6670	0.6366	0.6129	13462

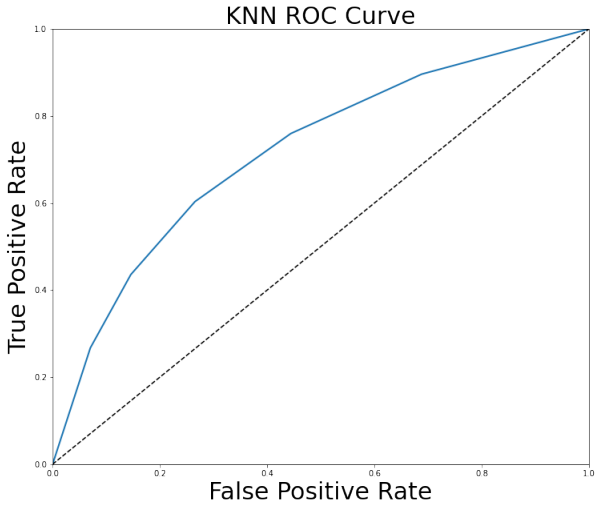

Figure 7: ROC Curve of the KNN Model

As we can see from the results, the KNN model is able to classify lower rated reviews relatively well, but seems to fall short when classifying highly-rated reviews. The overall accuracy is about 10% less than the expected accuracy from the cross validation. Therefore, while KNN is a highly flexible model and recommended for large datasets, there are still shortcomings. In addition, for a high dimension dataset like our dataset, it can be very computationally expensive to calculate distances for every data point. For these reasons, it is imperative that we consider other models as well.

3.3 Linear and Quadratic Determinant Analysis

The models that we have looked at so far have all been discriminant models. Discriminant models capture the conditional probability of the response variable given the data. We would now like to explore Generative models, which capture the joint probability of the response variable and predictors. Two such generative models are Linear Discrimi-

nant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) [5].

For both LDA and QDA, we make the assumption that the conditional distribution of the predictors given the response variable follows multivariate normal distribution. The difference between LDA and QDA is that LDA assumes that covariance matrix of the predictors is the same for both classes whereas QDA does not make this assumption.

LDA and QDA are quite attractive as there are no hyperparamters to tune. We fit both LDA and QDA models to the training set and received the following results on the test set:

Table 6: LDA Results

	Precision	Recall	F1-Score	Support
False	0.5149	0.7881	0.6228	3408
True	0.9124	0.7483	0.8222	10054
Accuracy	–	–	0.7584	13462
Macro Avg	0.7136	0.7682	0.7225	13462
Weighted Avg	0.8118	0.7584	0.7718	13462

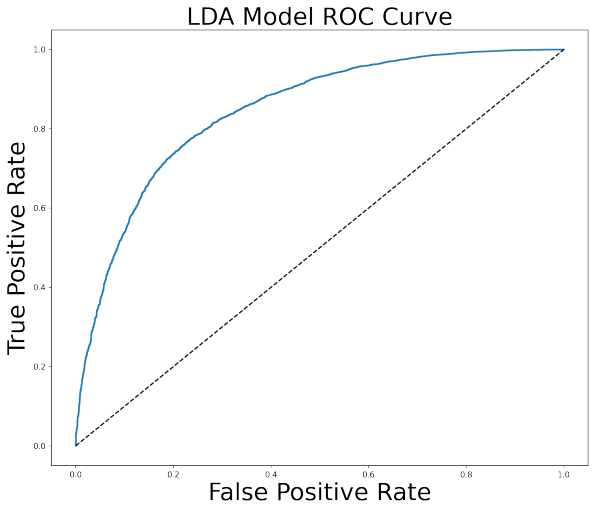

Figure 8: ROC Curve of the LDA Model

Table 7: QDA Results

	Precision	Recall	F1-Score	Support
False	0.3888	0.7306	0.5075	3408
True	0.8699	0.6107	0.7176	10054
Accuracy	–	–	0.6411	13462
Macro Avg	0.6294	0.6707	0.6126	13462
Weighted Avg	0.7481	0.6411	0.6644	13462

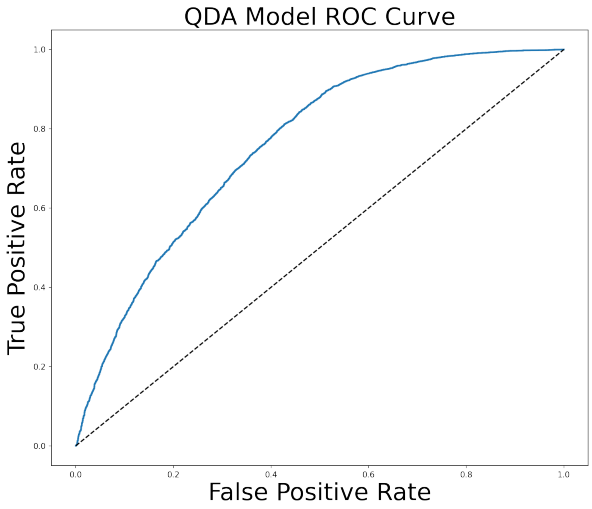

Figure 9: ROC Curve of the QDA Model

As we can see here, LDA has outperformed QDA in all the evaluation metrics. One explanation for this is that QDA is a more flexible model since LDA is a special case of QDA. LDA and Logistic Regression generally tend to perform similarly and that is the case with our models as well.

3.4 Random Forests

Decision trees are the backbone of random forest models [11], the last model we used to analyze the Yelp data. Decision trees work by splitting the training data into smaller and smaller subsets at nodes until each subset contains, ideally, only samples from one class. The tree and its nodes can then be used to make predictions on new data. Random forest models use a combination of many, differ-

ent decision trees, each trained using a different, random subset of predictors and training data. Then, the random forest model's final prediction is whichever class got the most votes from its individual decision trees. Because random forest models aggregate the predictions of many decision trees, they often have higher predictive power in comparison to other models.

We trained a random forest model to determine how well it could be used to predict Yelp ratings. Similar to our previous models, we trained the random forest model on the SMOTE balanced data with the 200 principal components from the PCA as the predictors and whether or not the review was 4 or 5 stars as our response variable.

Random Forest models have a large number of hyper-parameters than can impact the training process. In order to determine the best ones to use, we used Random Search Cross Validation and searched through 100 different combinations of hyper-parameters. The results showed that the ideal configuration of hyper-parameters was to not use bootstrapping, have a maximum depth of 90 for each tree, use square root to determine the maximum number of features for each tree, have a minimum number of 1 sample per leaf, have a minimum number of 2 samples for a split to occur, and to train 200 total trees.

It is important to note that one weakness of random forest models is that they take an incredibly large amount of processing to train. Thus, it was infeasible for us to conduct a Grid Search CV, and a Random Search CV was used instead. Even using this alternative method, it still took an extremely long time in order to train all the models and find the best combination of hyper-parameters. Utilizing the testing data, the performance of our final model and its ROC curve are displayed below.

Table 8: Random Forest Test Results

	Precision	Recall	F1-Score	Support
False	0.6352	0.5058	0.5632	3408
True	0.8433	0.9015	0.8714	10054
Accuracy	–	–	0.8014	13462
Macro Avg	0.7392	0.7037	0.7173	13462
Weighted Avg	0.7906	0.8013	0.7934	13462

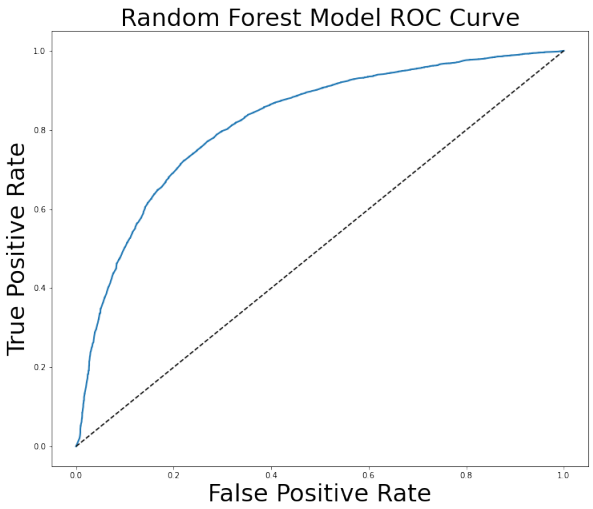

Figure 10: ROC Curve of the Random Forest Model

We see that, the F1-score is high for samples where the review was a 4 or 5, but still relatively low for reviews under 4 stars. Furthermore, the ROC curve is not as close to the top-left of the graph as one would hope. This result is likely a reflection of the difficulty of classifying with review data and the inability to fully optimize the model's hyper-parameters.

4 Conclusion and Summary

Here we conclude with our results and analysis, as well as some final remarks.

4.1 Results and Analysis

To analyze our models, we first compare their accuracy scores on our test dataset. Below we display each model's accuracy score, ordered from lowest to highest.

Table 9: Model Accuracy

	Accuracy
KNN	0.6366
QDA	0.6411
LDA	0.7584
LR	0.7712
RF	0.8041

We can see that our Random Forest model performed the best in terms of accuracy, with an accuracy of 80.41%. This makes sense as Random Forest models typically do very well at predictions since the model's addition of "randomness" allows for it to exploit its flexibility to fit the data well while at the same time reducing the common issue of overfitting to the data from being too flexible.

Following our Random Forest model, we see that our Logistic Regression and LDA models were the next best performers with accuracy scores of 77.11% and 75.84%, respectively. We don't believe that our reduced sentiment lexicon data had any Gaussian structure, so it makes sense that LDA and Logistic Regression performed relatively the same, with Logistic Regression slightly outperforming LDA.

What is interesting to note is that both our QDA and KNN models were the lowest performers with accuracy scores of 64.11% and 63.66%, respectively. This leads us to assume that because these models are typically more "flexible," our QDA and KNN models may have overfit to the training data too much, causing it to under-perform on the test set where the majority class dominates. Additionally, because our training set was balanced using SMOTE (over sampling from synthetically created datapoints), the minority class appeared too similar to each other, which in a way may encourage our more flexible models to overfit to those oversampled patterns.

To further understand how our models performed and validate our inferences, we next compare their performance on predicting ratings that were 4-5 stars (our "True"s), which is the majority class in our testing set. Below we display each model's precision, recall, and F1-scores for "True" responses, ordered from lowest to highest F1-score.

Table 10: Model "True" Classification Results

	T Precision	**T Recall**	**T F1-Score**
KNN	0.6034	0.8702	0.7128
QDA	0.8699	0.6107	0.7176
LDA	0.9124	0.7483	0.8222
LR	0.9073	0.7725	0.8345
RF	0.8433	0.9015	0.8714

We can see that LDA, followed by Logistic Regression, had the highest precision (91.24% and 90.73%, respectively), meaning most of its predictions of "True" were actually true. Thus their linear boundary from training was able to capture a sizable portion of the true values. KNN had the lowest precision at 60.34% but the second highest recall at 87.0%, indicating that it may be predicting most of the data points as "True" regardless of it they are actually "True" or not.

QDA had the lowest recall at 61.07% but third highest precision at 86.99%. Thus we believe our QDA model may have overfit a boundary around our training "True"s which ultimately allowed it to be more precise about what it labeled as true in the test set, but still

missed a lot of actual "True"s.

Random forest had the highest recall at 90.15% and the third highest precision at 84.33%, telling us that it must do a good job of generalizing the information it gained from the training data to classify "True"s. Additionally, this is why we can see it has the highest F1-Score, which is a balance of precision and recall.

Finally, we compare their performance on predicting ratings that were 1-3 stars (our "False"s), which is the minority class in our testing set. Below we display each model's precision, recall, and F1-scores for "False" responses, ordered from lowest to highest F1-score.

Table 11: Model "False" Classification Results

	F Precision	**F Recall**	**F F1-Score**
KNN	0.7344	0.3857	0.5058
QDA	0.3888	0.7306	0.5075
RF	0.6352	0.5058	0.5632
LDA	0.5149	0.7881	0.6228
LR	0.5334	0.7670	0.6292

Because the "False"s were our minority class, these results can give us the clearest idea of how our model performed in regards to accurately predicting the rating group of Yelp reviews.

KNN and Random Forest had the highest precision of 73.44% and 63.52%, respectively, but the lowest recall of 38.57% and 50.58%, respectively. As we noted below Table 8 on accuracy, since both KNN and Random Forest are very flexible models and our minority class was over sampled via SMOTE, these models must have fit well to the replicated minority class, which naturally made it overfit to very similar samples and thus lead to high precision but an inability to classify all the "False"s as "False."

We see the opposite case for QDA where its precision was the lowest at 38.88% while its recall was one of the highest at 73.06%. As we suspected from the previous note on QDA's "True" precision and recall, our QDA model must have overfit a boundary around our training "True"s and thus led to this affect where it's "True" precision is high while recall is low and "False" precision is low where recall is high. Overall, it just doesn't do a good job.

Due to these clear issues with flexibility leading to overfiting, our linear boundary models Logistic Regression and LDA achieve the highest F1-scores by having not-the-best precision (53.34% and 51.49%, respectively) but achieving the greatest recall (76.7% and 78.81%, respectively). We can see that despite the minority class training data being oversampled, these models were able to avoid overfitting to the data because of their inherent inflexibility and thus identified most of the "False"s in our test set as "False".

Overall, while our Random Forest had the highest accuracy, we can see from our "True" and "False" precision, recall, and F1-scores that it was actually Logistic Regression and LDA that performed the best at accurately classifying both the 1-3 star ratings and 4-5 star ratings. In the real world there will naturally be more 4-5 star ratings (inferred from the original balance of our dataset), so if one is interested in just overall accuracy (since this is not a life-or-death situation) then a Random Forest may be the best choice. However, if one is more concerned about getting the more accurate results for each group of ratings, then using a model with less flexibility, such as a Logistic Regression model or an LDA model, may be the best choice.

4.2 Final Remarks

While our models were above the threshold of 50%, meaning they doing better than randomly guessing, we still believe there are areas of opportunities we could address looking forward to improve our models.

Firstly, our data may not be independent (a necessary assumption when modeling our data). This is because new reviews may be influenced by older reviews, as users see reviews before going to a restaurant, and that influences their perception of the food/restaurant and ultimately the review they write. Additionally, because we used SMOTE, we had a lot of synthetically replicated samples in our training set which may have caused our models to overfit to the minority class instead of capturing the true ranges the minority class's predictors may exhibit.

Secondly, there may be some inherent issues with our preprocessing method. We employed a simple text analysis that only counts frequencies of words appearing in a sentence. There is no indication of the surrounding context of those "positive" and "negative" words, which may alter our understanding of if those words were truly "positive" or "negative" based on the context (i.e. "It was not good" and "It was good" may just extract "good" and be given the same value). For PCA, we subjectively chose a threshold of 80% for dimension reduction when conducting our PCA, which may not be an adequate cumulative variance needed for our prediction problem (especially since some top components we selected may just contain high variance from noise). If we were only using one model for prediction, we could use K-fold CV to select the best number of components to include from PCA.

References

[1] Yelp dataset. *Yelp*, 2014. `https://www.yelp.com/dataset`, Accessed 20-November-2022.

[2] Performance measures: Cohen's kappa statistic. *The Data Scientist*, 2021. `https://thedatascientist.com/performance-measures-cohens-kappa-statistic/`, Accessed 24-November-2022.

[3] J. Brownlee. SMOTE for imbalanced classification with python. *Machine Learning Mastery*, 2020. `https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/`, Accessed 22-November-2022.

[4] T. Edgar and D. Manz. Logistic regression. *ScienceDirect*, 2017. `https://www.sciencedirect.com/topics/computer-science/logistic-regression`, Accessed 22-November-2022.

[5] B. Ghojogh and M. Crowley. Linear and quadratic discriminant analysis. *ARXIV*, 2019. `https://arxiv.org/abs/1906.02590`, Accessed 22-November-2022.

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani. An introduction to statistical learning. *New York: Springer Science Business Media*, 2013.

[7] A. Kumar. PCA explained variance concepts with python example. *Vitalflux*, 2022. `https://vitalflux.com/pca-explained-variance-concept-python-example/`, Accessed 22-November-2022.

[8] S. Satpathy. Overcoming class imbalance using SMOTE techniques. *Analytics Vidhya*, 2020. https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/, Accessed 22-November-2022.

[9] P. Sharma. The most comprehensive guide to K-means clustering you'll ever need. *Analytics Vidhya*, 2019. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/, Accessed 29-November-2022.

[10] T. Srivastava. Introduction to K-nearest neighbors: A powerful machine learning algorithm. *Analytics Vidhya*, 2018. https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, Accessed 22-November-2022.

[11] E. Sruthi. Understanding random forest. *Analytics Vidhya*, 2021. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest//, Accessed 22-November-2022.