Christian Becerra
Nicholas Hansen
Kheva Mann
CSC 369-03

Final Project: Yelp Review Rating Predictions

Overview

For our project, we aimed to develop a classification model that could accurately predict the ratings given by users to businesses based on their reviews. To do this, we used a publicly available dataset of Yelp business reviews that has been constructed for natural language processing (NLP) and other various machine learning tasks. Each record in this dataset contains a lot of metadata about the review including the text content and an associated categorical rating ranging from 1 to 5. We processed the reviews using Apache Spark on a distributed Hadoop file system, converting them into term frequency-inverse document frequency (TF-IDF) vectors and calculating their cosine similarity. We then ran both regressive and classification K-nearest neighbor algorithms on these similarities in order to make predictions about the ratings. While this process can be considered a rudimentary form of sentiment analysis, it is limited by its reliance on pure word frequencies and does not take into account the context of individual words. This contrasts with more complex NLP techniques that can consider such factors in their analysis.

Data Preprocessing

The Yelp review dataset, which is approximately 6GB in size, was too large to analyze in its entirety. Therefore, we selected subsets of the data ranging from 500 to 1200 reviews per categorical rating for our analysis. This reduction in sample size was necessary due to the exponential growth in the number of similarity calculations required with increasing corpus size. Furthermore, this subset selection ensured a balanced dataset and avoided bias. In order to prepare the data for analysis, we extracted the rating and text for each review record, assigning a unique identifier to each record. The text of the reviews was then preprocessed by removing punctuation, extra spacing, and stop words. Stop words are common words that are frequently present in text but do not provide significant meaning, such as articles and pronouns. However, we retained stop words that were relevant for sentiment analysis, such as "good" or “bad”. These words were removed in order to reduce noise in the analysis and improve the accuracy of the model. Further preprocessing could be done such as word stemming (reduce “running” to “run”) and contraction expansion (expand “they’re” to “they are”) to improve TF-IDF vector quality, however this was determined as outside the scope of this project.

TF-IDF Implementation

In order to calculate the similarity between documents, the text of the documents must be converted into numerical vectors. One common way to do this is to use term frequency-inverse document frequency (TF-IDF) vectors. TF-IDF is a statistical measure that reflects how important a word is to a document in a corpus which is calculated which are calculated using the following formula:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) * \text{IDF}(t, D)$$

where t is a term, d is a document, and D is a corpus of documents. The term frequency is the number of times a word appears in a document, and the inverse document frequency is the logarithm of the total number of documents divided by the number of documents containing the word.

The resulting vectors can then be used to calculate the similarity between documents using distance measures such as cosine similarity. In the case of the K nearest neighbors algorithm, these similarity vectors are used to find the k documents that are most similar to a given document. This allows the algorithm to make predictions based on the ratings of the most similar documents. In our analysis, we created the TF-IDF vectors by considering only the 1000 most prevalent words in the corpus, as these words are more likely to indicate sentiment than less frequent words, such as business names and food items. The vectors were then converted to sparse vectors using the Spark MLlib library to save space and improve computational efficiency.

KNN Implementation

In order to calculate the cosine similarity between all pairs of documents in a corpus, we first normalized the sparse TF-IDF vectors based upon their magnitudes allowing us to compare the vectors on an equal footing regardless of their original magnitudes.

Next, we computed the dot product between all pairs of documents using the cartesian product of all the vectors. This is equivalent to calculating the cosine similarity between the documents as the dot product is directly related to the cosine of the angle between the vectors. To ensure that our calculations were not influenced by irrelevant or redundant information, we filtered out situations where a document was compared to itself. This involved checking each pair of documents to see if they were identical, and discarding any pairs that were.

We then employed the KNN algorithm to make predictions or recommendations based on the k documents with the highest similarity values to each document. There are two common ways of

using KNN for this purpose: regressive KNN and classifier KNN. In regressive KNN, we use the average rating of the k nearest neighbors to make a prediction for a given query document. This is useful when the target variable is continuous, such as a rating or a price. In contrast, classifier KNN uses the plurality rating of the k nearest neighbors to classify a query document into one of several predefined classes. This is useful when the target variable is categorical, such as a class label. We opted to test our solution with both regressive and classifier KNN algorithms as although our ground truth label, “rating”, is often in a continuous form, the Yelp dataset only provides this information categorically.

To determine the value of k to use in our KNN calculations, we employed a commonly used heuristic and set k equal to the square root of the number of documents in the corpus. This ensured that we included a sufficient number of nearest neighbors to make accurate predictions or recommendations.

Results

All of the results below are created utilizing a constant TF-IDF vector size composed of the top 1000 most frequent words in the review corpus. Limiting the TF-IDF vectors to these words was not only done for the sake of program efficiency, but also to limit the influence of less common or unique words that are the least likely to indicate sentiment and the most likely to influence document separability. These words could be things such as names of businesses or specific food items that would be important for determining business and authorship attribution, but are not relevant for determining sentiment.

Both KNN regression and categorical classification were tested for predicting review ratings, however since the ground truth review labelings in the Yelp dataset are categorical (take on only the values 1.0, 2.0, 3.0, 4.0 and 5.0), categorical classification tended to perform better on average.

A constant number of reviews per rating were considered in the corpus to ensure a balanced dataset. In addition, 500, 1000, and 1200 reviews per rating were tested corresponding to 2500, 5000, and 7000 review corpus sizes respectively. The value of K in the KNN algorithm was set at a standard value equal to the square root of the review corpus size.

After conducting analysis on our models, we found that the two best performing models used 1200 and 1000 documents per rating, achieving an accuracy of 45.55% and 45.52% in classification KNN, respectively. The root mean squared error for these models was 1.132 stars and 1.367 stars in regressive KNN, respectively. While the models were similar in performance, exploring classifier KNN metrics reveals that the 1000 document model performed slightly

better in predicting 2 star ratings whereas the 1200 document model performed slightly better in predicting 4 star ratings.

When comparing classifier KNN and regressive KNN, classifier KNN generally performed better in predicting ground truth ratings at the label bounds (i.e. 1 star and 5 stars) and adequately predicted ratings at the label center (2-4 stars). Regressive KNN, on the other hand, performed well in predicting ratings at the center but struggled in predicting ratings near the bounds. This may be due to the lack of intermediate values (i.e. 1.5 stars) which can cause regression averaging to skew towards the middle values and away from the bounds.

Overall, we are pleased with these results considering that the predictions were based solely on word frequency information from TF-IDF vectors and did not incorporate contextual information like more advanced NLP methodologies.

500 Reviews per Rating

Classification

Confusion Matrix - Predictions in Columns

1.0	345	78.0	30.0	27.0	20.0
2.0	176.0	131.0	81.0	71.0	41.0
3.0	86.0	92.0	123.0	120.0	79.0
4.0	47.0	63.0	72.0	164.0	154.0
5.0	38.0	21.0	28.0	127.0	286.0
	1.0	**2.0**	**3.0**	**4.0**	**5.0**

Overall Accuracy: 0.4196

Precision By Label

Precision(1.0) = 0.4985549132947977
Precision(2.0) = 0.34025974025974026
Precision(3.0) = 0.36826347305389223
Precision(4.0) = 0.32220039292730845
Precision(5.0) = 0.49310344827586206

Recall By Label

Recall(1.0) = 0.69
Recall(2.0) = 0.262
Recall(3.0) = 0.246
Recall(4.0) = 0.328
Recall(5.0) = 0.572

F-Measure By Label

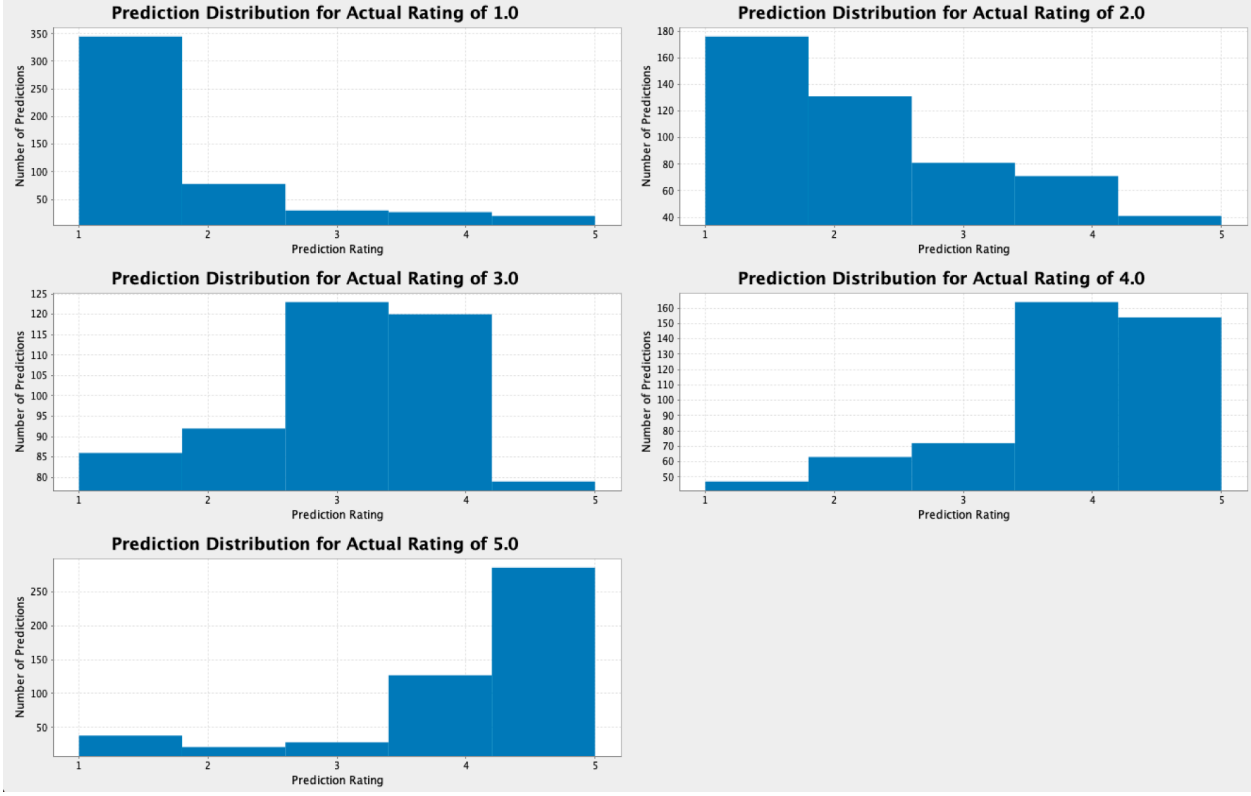
F1-Score(1.0) = 0.5788590604026844

F1-Score(2.0) = 0.296045197740113

F1-Score(3.0) = 0.2949640287769784

F1-Score(4.0) = 0.3250743310208127

F1-Score(5.0) = 0.5296296296296297



Regression

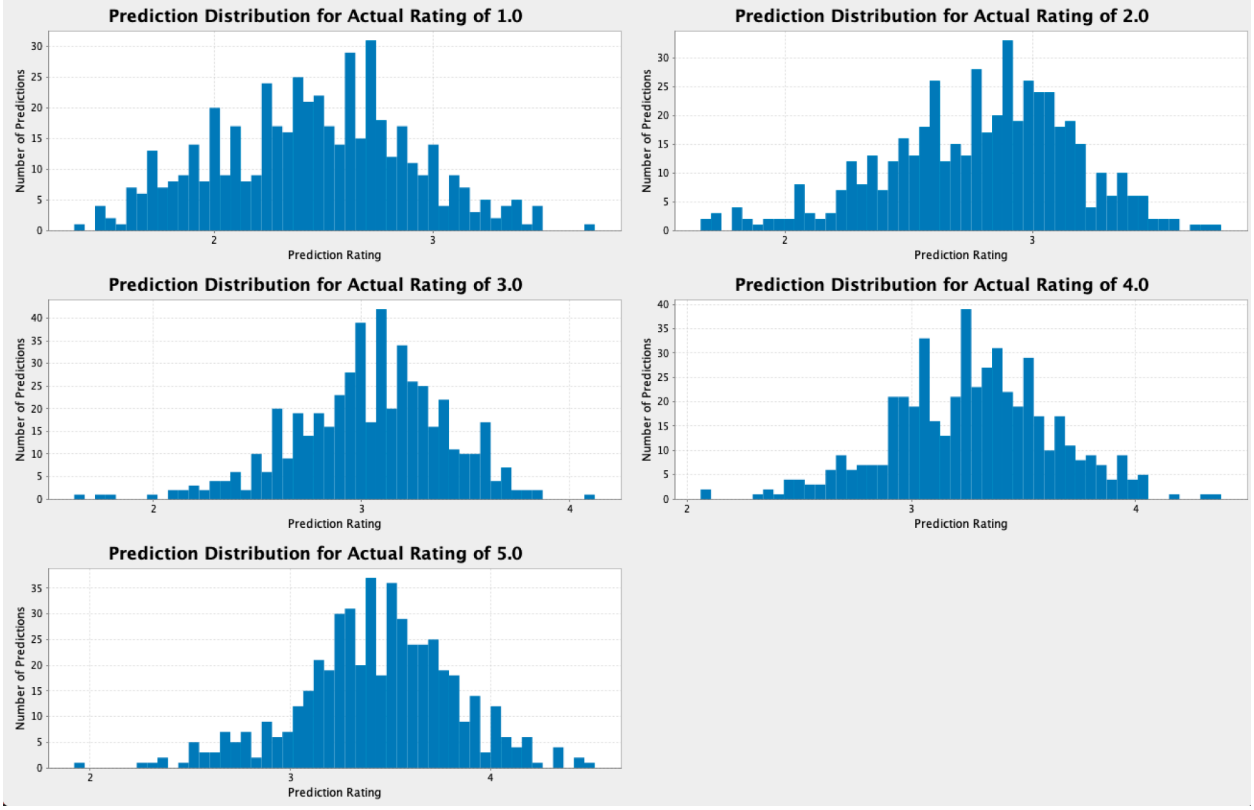
MSE = 1.2948835200000004

RMSE = 1.1379294881494197

R-Squared = 0.35255824000000036

MAE = 0.9680639999999993

Explained Variance = 0.27094751999999966



1000 Records per Rating

Classification

Confusion Matrix - Predictions in Columns

1.0	666.0	171.0	54.0	62.0	47.0
2.0	292.0	339.0	159.0	141.0	69.0
3.0	145.0	205.0	290.0	229.0	131.0
4.0	66.0	102.0	191.0	328.0	313.0
5.0	71.0	45.0	60.0	221.0	603.0
	1.0	**2.0**	**3.0**	**4.0**	**5.0**

Overall Accuracy: 0.4452

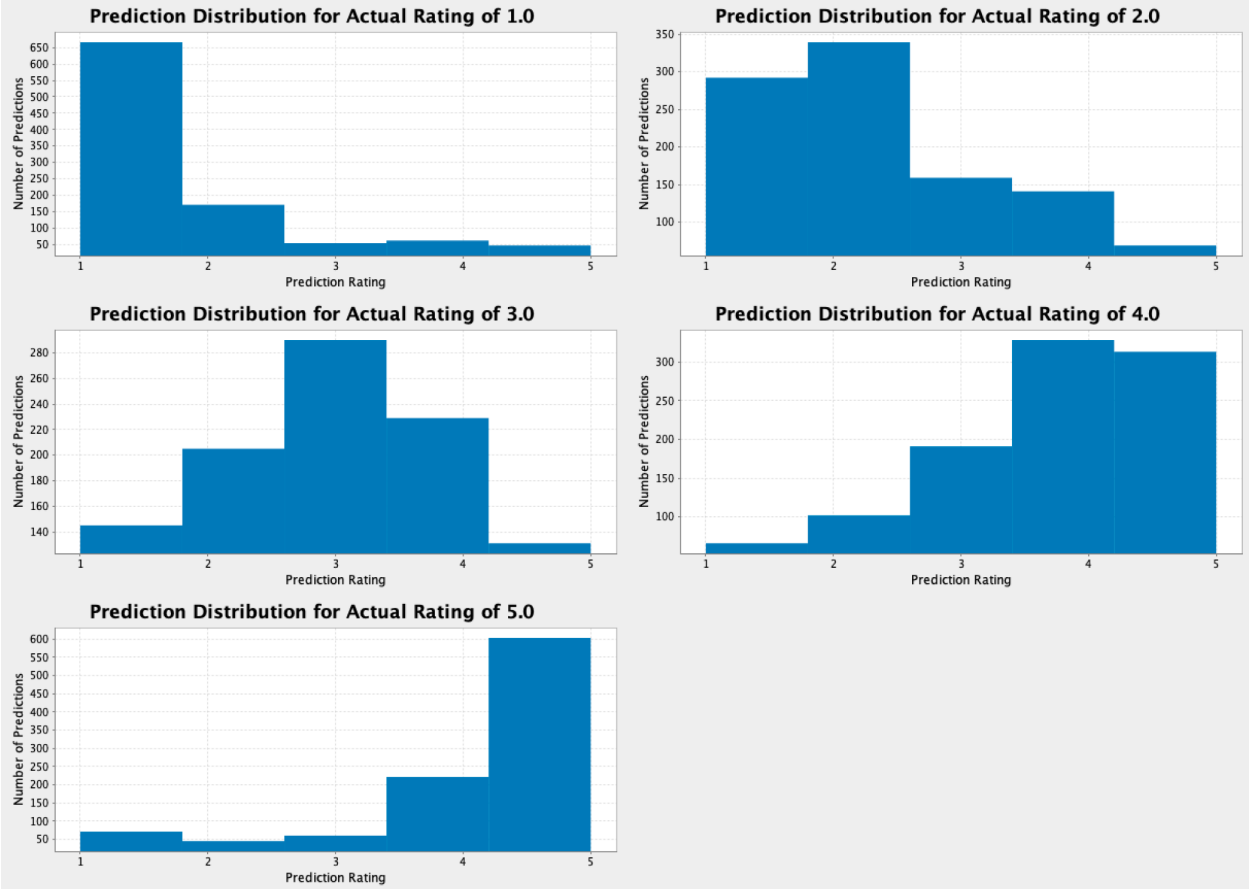
Precision By Label

Precision(1.0) = 0.5370967741935484
Precision(2.0) = 0.39327146171693733
Precision(3.0) = 0.38461538461538464
Precision(4.0) = 0.3343527013251784
Precision(5.0) = 0.5184866723989682

Recall By Label

Recall(1.0) = 0.666
Recall(2.0) = 0.339
Recall(3.0) = 0.29
Recall(4.0) = 0.328
Recall(5.0) = 0.603

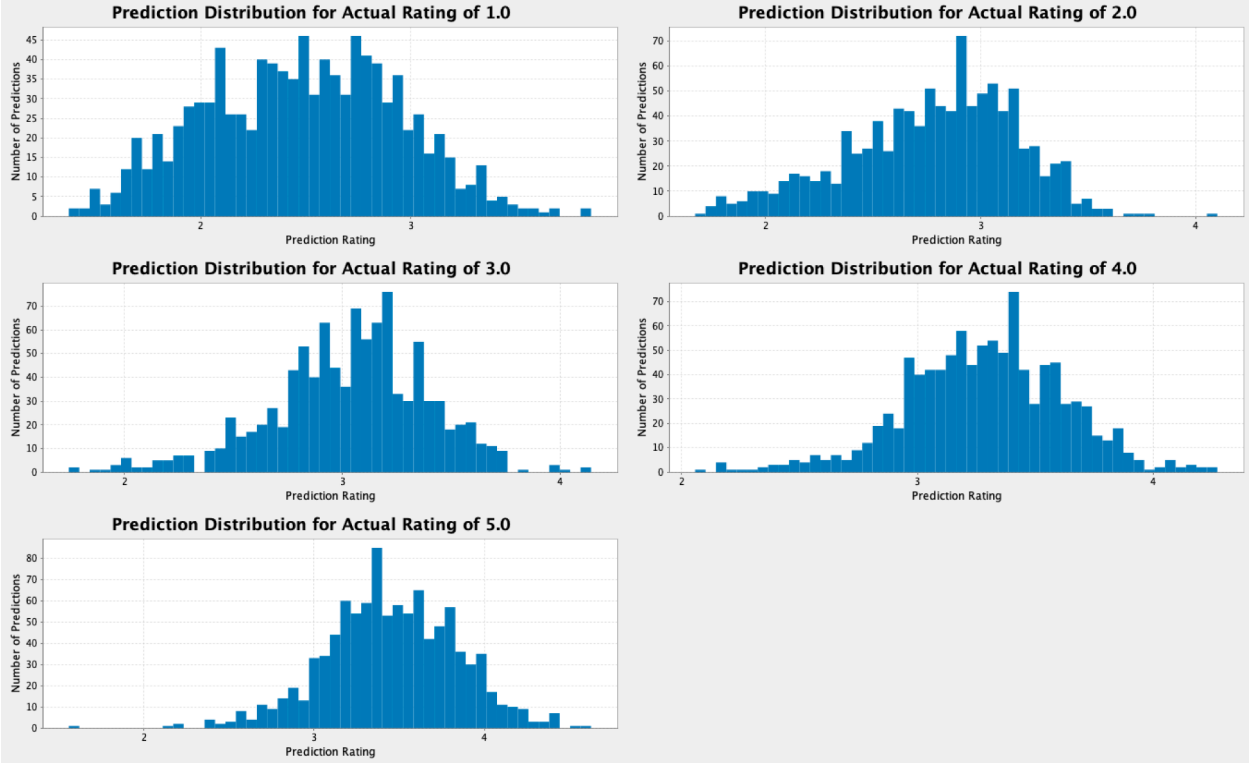
F-Measure By Label
F1-Score(1.0) = 0.5946428571428573
F1-Score(2.0) = 0.36412459720730395
F1-Score(3.0) = 0.330672748004561
F1-Score(4.0) = 0.331145885916204
F1-Score(5.0) = 0.5575589459084604



Regression

MSE = 1.2919737142857146
RMSE = 1.1366502163311785
R-Squared = 0.35401314285714225
MAE = 0.9638399999999999

Explained Variance = 0.273030857142857



1200 Records per Rating

Classification

Confusion Matrix - Predictions in Columns

1.0	816.0	195.0	59.0	69.0	61.0
2.0	383.0	371.0	214.0	152.0	80.0
3.0	172.0	234.0	352.0	298.0	144.0
4.0	86.0	107.0	223.0	424.0	360.0
5.0	87.0	50.0	78.0	275.0	710.0
	1.0	**2.0**	**3.0**	**4.0**	**5.0**

Overall Accuracy: 0.4455

Precision By Label

Precision(1.0) = 0.5284974093264249
Precision(2.0) = 0.3876698014629049
Precision(3.0) = 0.3801295896328294
Precision(4.0) = 0.34811165845648606
Precision(5.0) = 0.5239852398523985

Recall By Label

Recall(1.0) = 0.68

Recall(2.0) = 0.30916666666666665

Recall(3.0) = 0.29333333333333333

Recall(4.0) = 0.35333333333333333

Recall(5.0) = 0.5916666666666667

F-Measure By Label

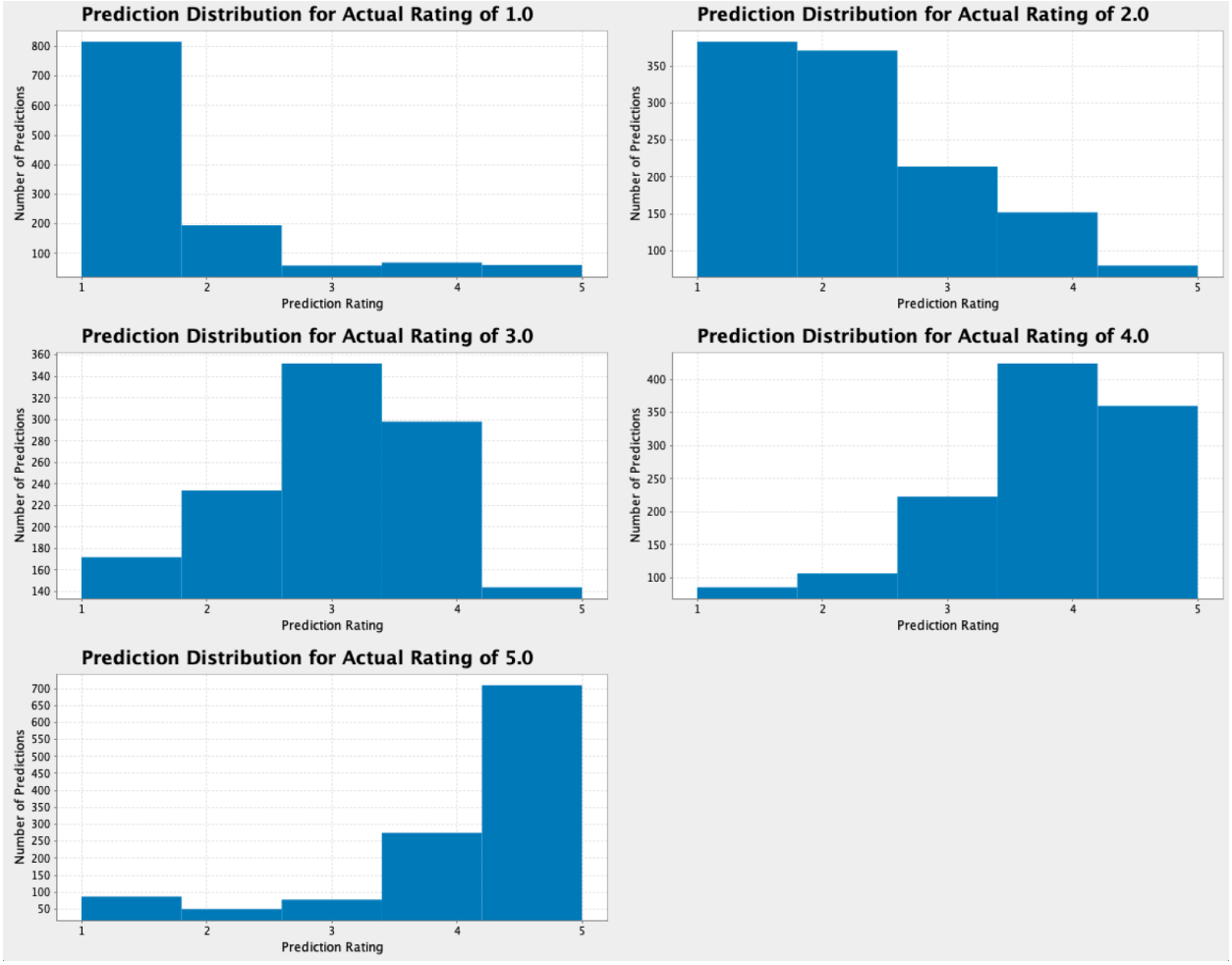
F1-Score(1.0) = 0.5947521865889213

F1-Score(2.0) = 0.34399629114510893

F1-Score(3.0) = 0.33113828786453436

F1-Score(4.0) = 0.35070306038047977

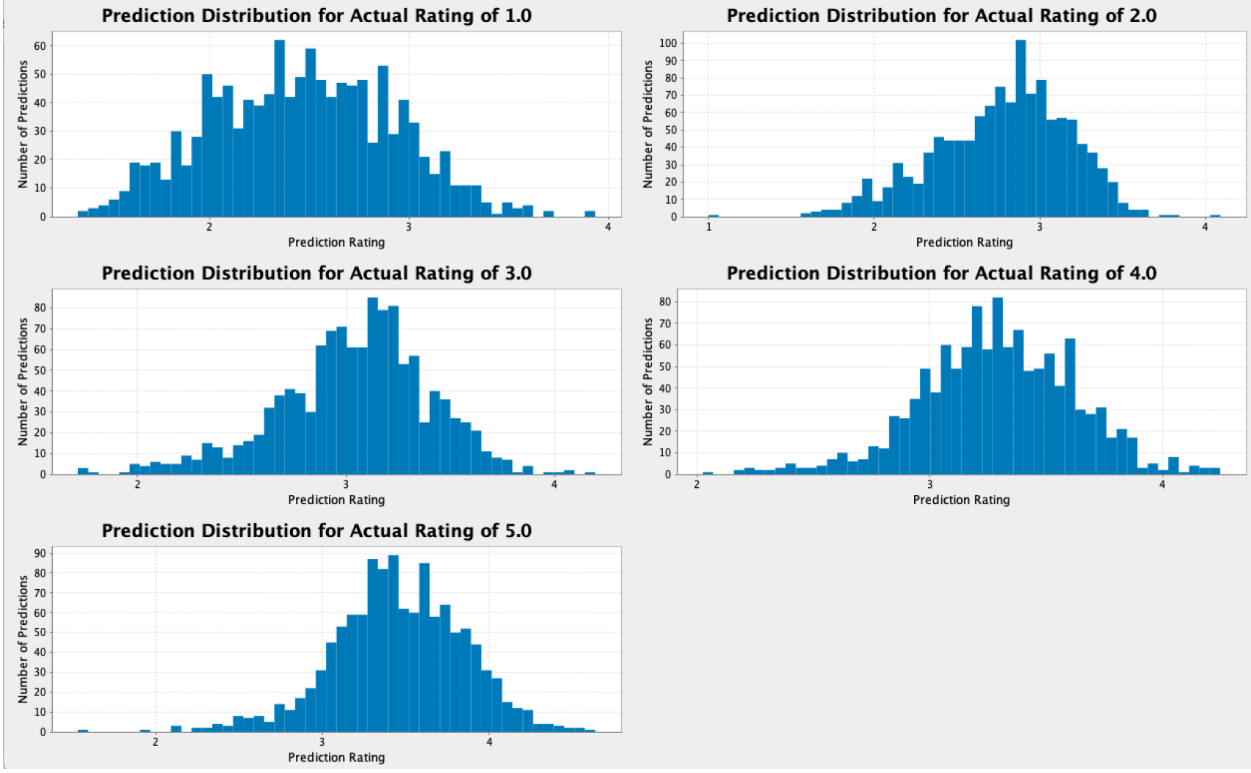
F1-Score(5.0) = 0.5557729941291585



Regression

MSE = 1.281426519368078

RMSE = 1.1320011127945406
R-Squared = 0.359286740315961
MAE = 0.9595822510822511
Explained Variance = 0.2805044414459998



Limitations & Future Work

Overall we were able to get some interesting results, however our model is far from perfect. Refining our model goes a bit beyond the scope of this class, however we did compare outputs from modifying the number of documents per rating as well as the number of top words in our TF-IDF vector. We ultimately settled on taking a random sample of 1200 reviews for each rating (1-5) and considered the 1000 most common words when creating our vectors. We settled on these numbers by finding a sweet spot between the accuracy of our model and the time it takes to process the data. With faster computers or a larger cluster it would be beneficial to try and increase the number of reviews we could process as well as comparing more words in our vectors. Another improvement that could be made to refining our model is to experiment with more K values for our KNN algorithm. We set a default to the square root of our documents, however the accuracy may improve by tweaking that value. Finally, vast improvements could be made by leveraging more advanced NLP techniques that consider word context (i.e. “not good”) for prediction rather than pure term frequency.