Predicting Review Ratings from Yelp Dataset

Christian Becerra
Nicholas Hansen
Kheva Mann

Problem Statement

- Want to predict the rating of reviews
- Predict ratings given by users to businesses based on reviews most texually similar to it
- Ratings are given on a 1 to 5 scale
- Reviews come from a very large (6 GB) dataset from the company Yelp
- Publicly available and intended for academic use

Solution

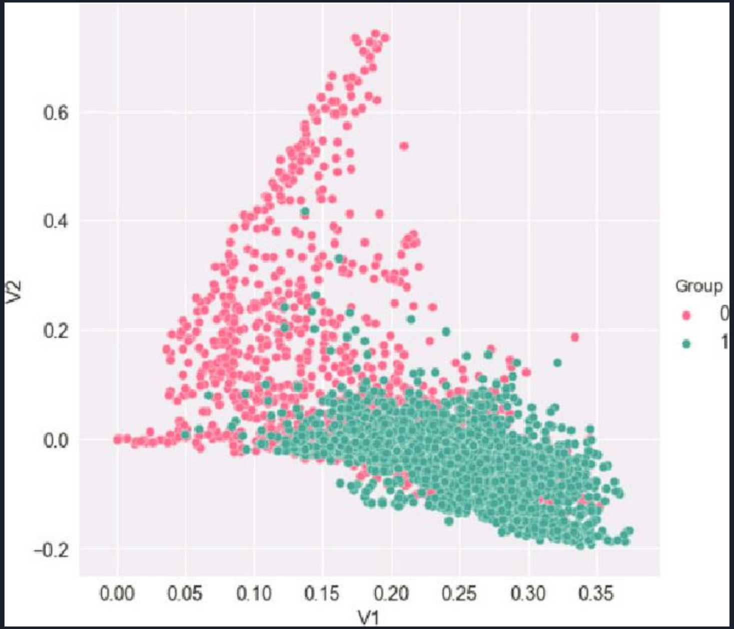
- Use Natural Language Processing with the Spark framework on the Hadoop FS
- Reviews are converted into term frequency-inverse document frequency vectors or TF-IDF
- Cosine similarity is then detected
- K-nearest neighbor algorithms allow us to determine our final prediction
- This technique is fairly simple but sufficient for our needs
- Definitely has room for improvement

Too much data

- The initial data set from Yelp is approximately 6 GB
  - Much too large, especially for creating TF-IDF vectors
- Data must be reduced
  - Only a subset of data was taken (less than 10 MB)
  - Remaining data was filtered to remove unnecessary information
  - Calculating similarity amongst all documents explodes data size to n^2

Cleaning and Vectorizing the Text

- Review text must be vectorized (made numerical) in order to calculate similarity between documents
- Don’t want noise in our data
    - Remove common stop words like “a” and “the” that provide little information (modified to keep common sentiment words)
    - Remove punctuation and extra spacing
    - Did not stem or expand contractions (possible improvement)



https://www.researchgate.net/figure/Dimensional-reduction-of-term-frequency-inverse-document-frequency-TF-IDF_fig2_330121599

TF-IDF Implementation

- TF-IDF determines how significant a word presence in a document is based upon the frequency of that word in the corpus.
    - Good for determining document uniqueness and clustering
    - TF-IDF(t, d, D) = TF(t, d) * IDF(t, D)
        - t = term
        - d = document
        - D = corpus of documents
- Considered top 1000 most prevalent words (better for sentiment)
- Used Spark MLib sparse vectors to represent (TF-IDF vectors are cluttered with 0s)

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

https://medium.com/@imamun/creating-a-tf-idf-in-python-e43f05e4d424

TF-IDF Implementation (Continued)

- TF-IDF normalized based on magnitudes allowing for fair comparisons
- Dot product is computed between all pairs of normalized document TF-IDF vectors (except self comparisons)
    - Equivalent to cosine similarity
    - Used to find most similar documents

KNN Implementation

- Regressive vs Classifier
  - Regressive
    - Average rating of the k nearest neighbors to make prediction
    - Useful for continuous variables (value predictions $)
  - Classifier
    - Plurality rating of the k nearest neighbors classifies variable into predefined class
    - Useful for categorical variables (interest predictions)
- Did both!

Results (Methods)

- Tested with 500, 1000, and 1200 documents per rating respectively
- 500 reviews per rating worst performer
- Constant 1000 top words TF-IDF vector size
- Tested both regressive and classifier KNN
    - K = sqrt(total_documents)
        - Standard K value

Results (1000 vs 1200) - Classifier

- Performed relatively similar
  - 1000 better at predicting 2.0 rating
  - 1200 better at predicting 4.0 rating
  - Likely transient rather than improved performance

1000 Reviews Per Rating

Overall Accuracy: 44.52%

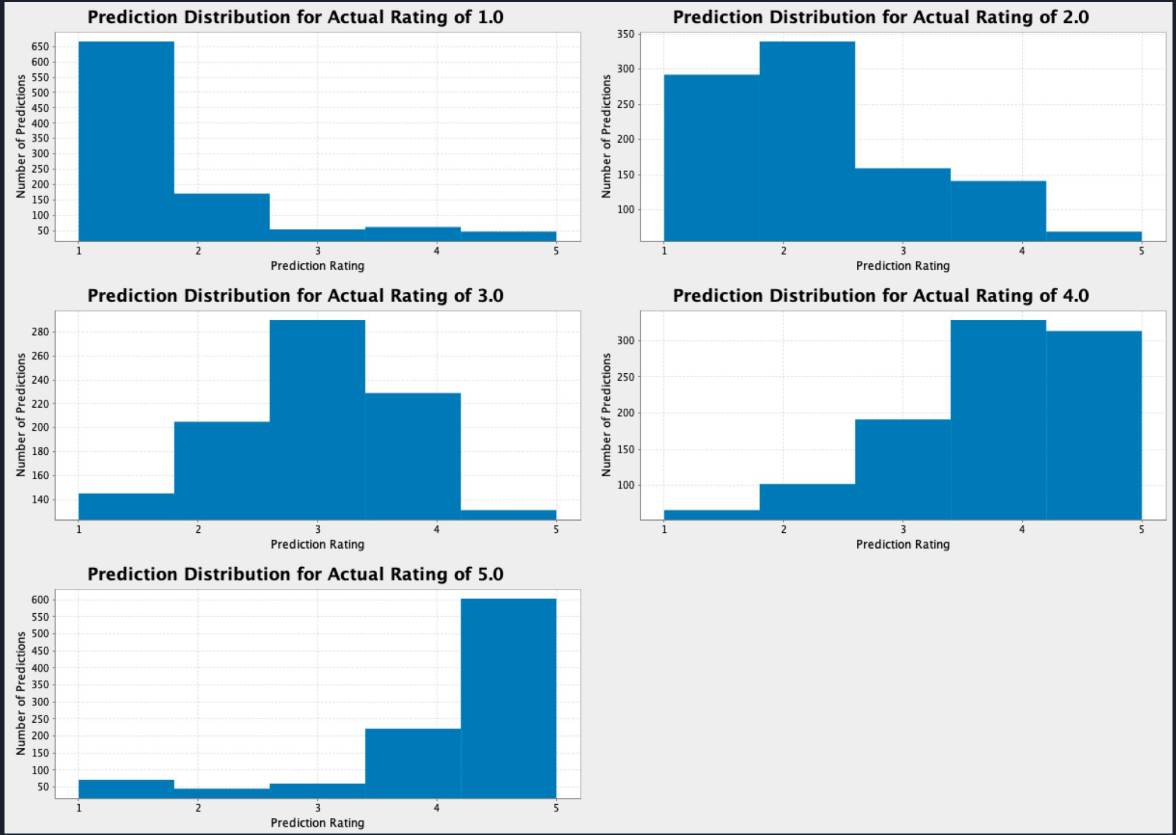
F1-Score(1.0) = 0.5946428571428573
F1-Score(2.0) = 0.36412459720730395
F1-Score(3.0) = 0.330672748004561
F1-Score(4.0) = 0.331145885916204
F1-Score(5.0) = 0.5575589459084604

1200 Reviews Per Rating

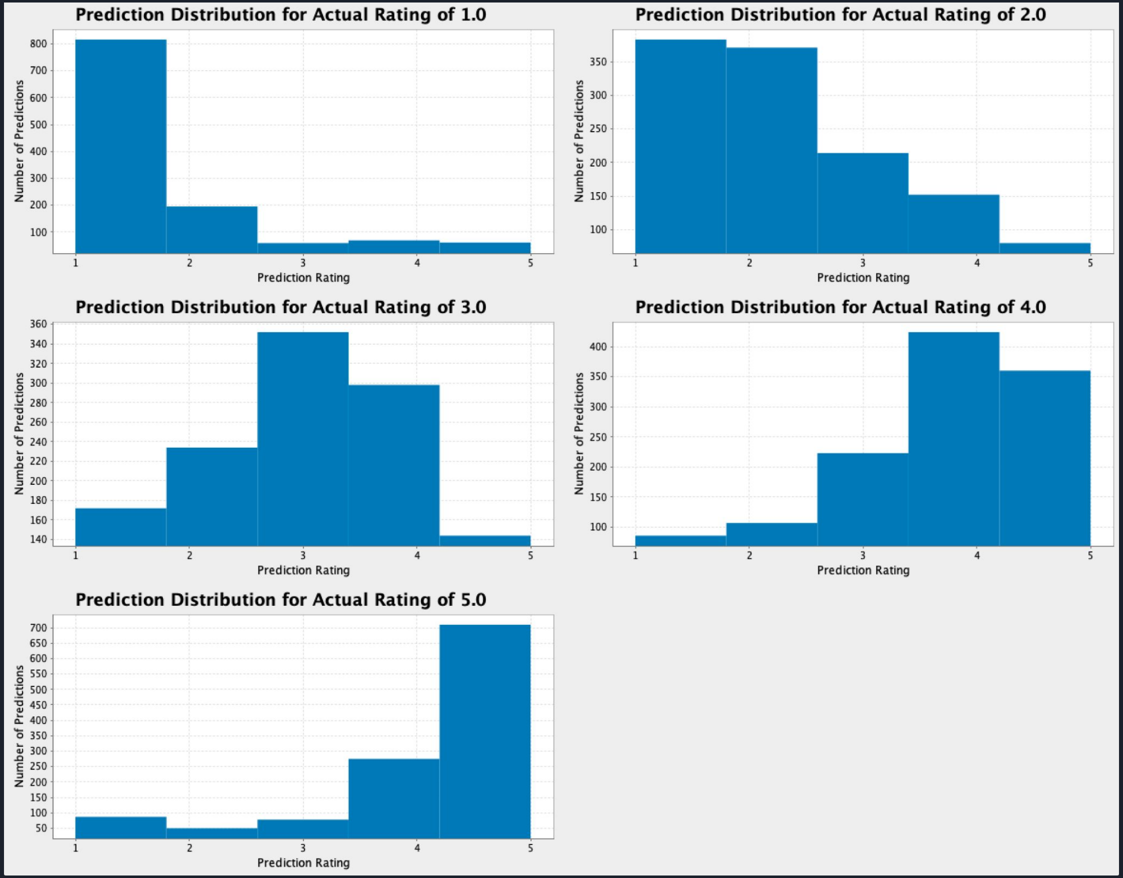
Overall Accuracy: 44.55%

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

Results (1000 per Rating) - Classifier



Results (1200 per Rating) - Classifier

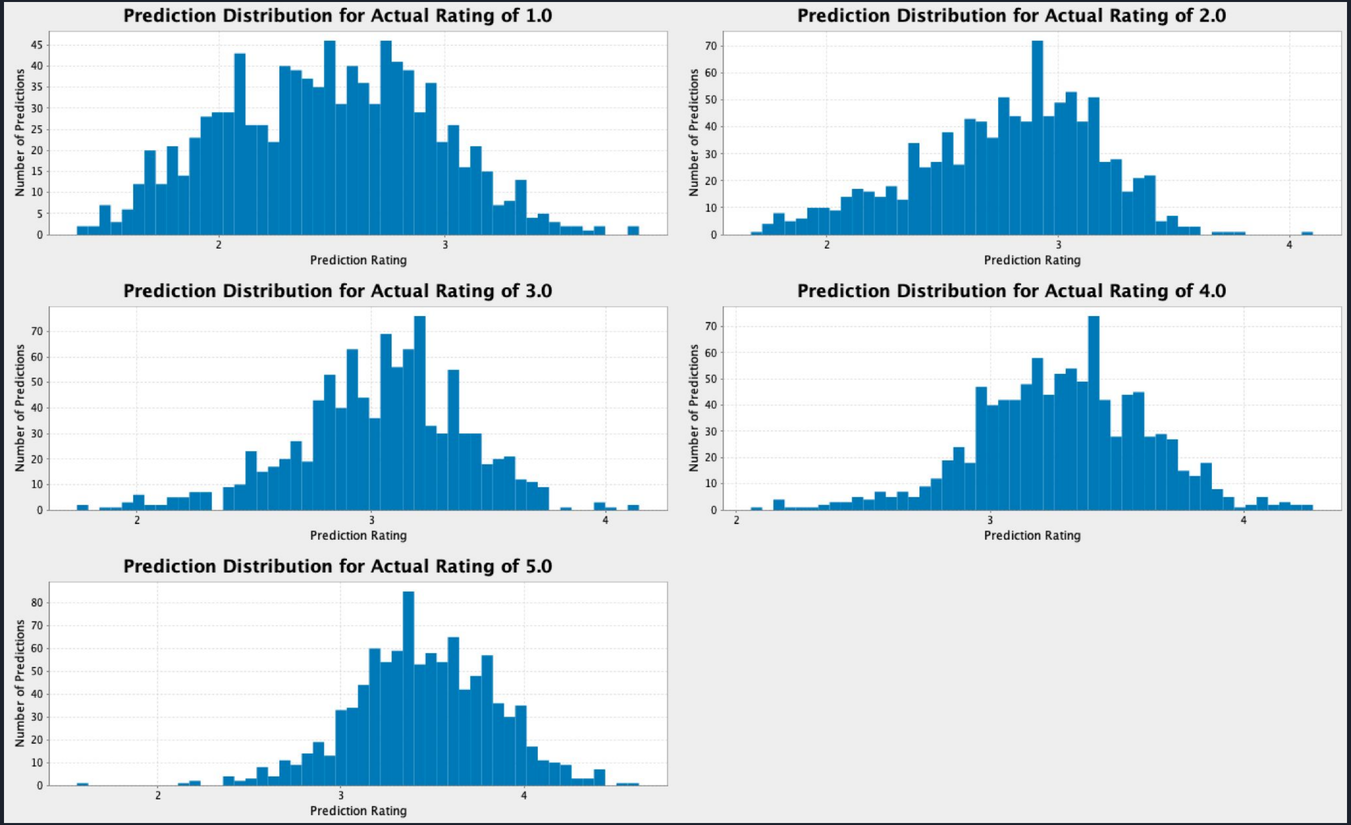


Results (1000 vs 1200) - Regressor

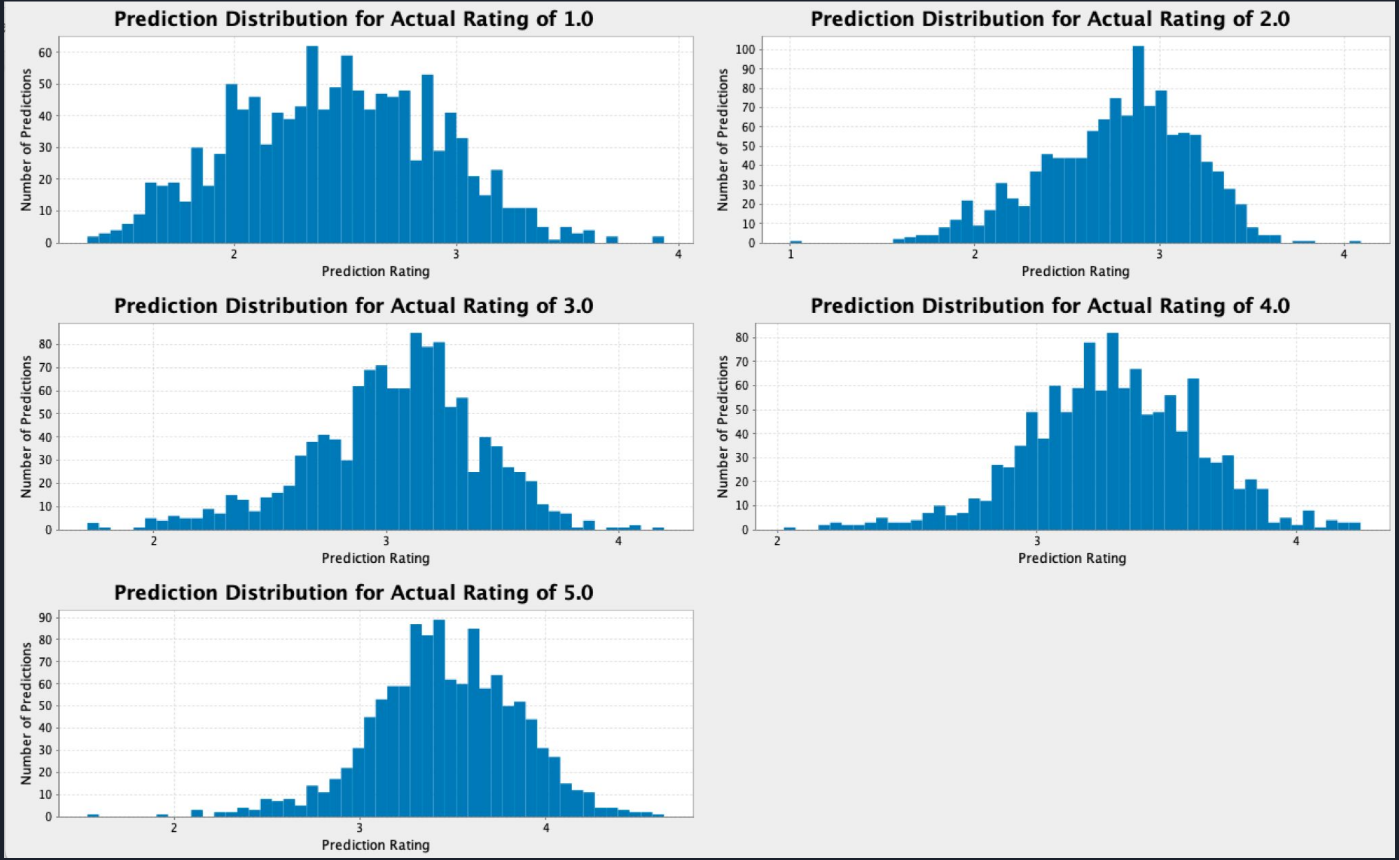
- Relatively the same

1000 Reviews Per Rating	1200 Reviews Per Rating
RMSE = 1.137 stars	RMSE = 1.132 stars
MAE = 0.964 stars	MAE = 0.96 stars

Results (1000 per Rating) - Regressor

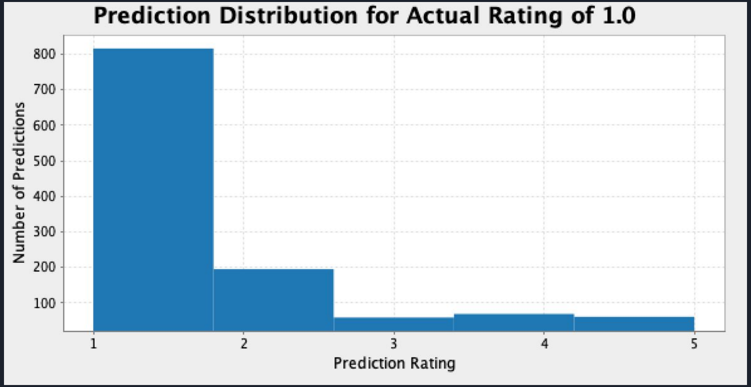


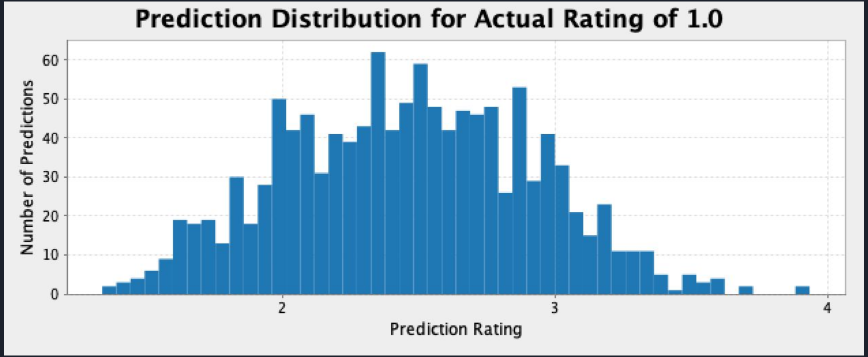
Results (1200 per Rating) - Regressor



Results: Classifier vs Regressor KNN

- Classifier good at predicting values at the bounds (1 star and 5 stars)
    - Decent at predicting values in the center (2-4 stars)
- Regressor decent at predicting values in the center
    - Bad at predicting values at the bounds
    - Taking the average biases values towards the center





What We Learned

- Vectorizing large amounts of data taking too long
    - Had to use Spark Mllib library for Sparse Vectors
- Learned a lot about text vectorization
- Learned about natural language processing and sentiment analysis
- Learned about KNN and two different ways of applying it
- Data visualization using MLlibrary
- Rudimentary due to methodology used compared to other strategies

Thank You!