Qualitate – Data Driven Restaurant Consulting

GitHub Repo

https://github.com/muditgoyal/yelp

Problem

Yelp is a laundry list of data points from customers such as checkins and business data. How might we help businesses improve their services through data found in reviews?

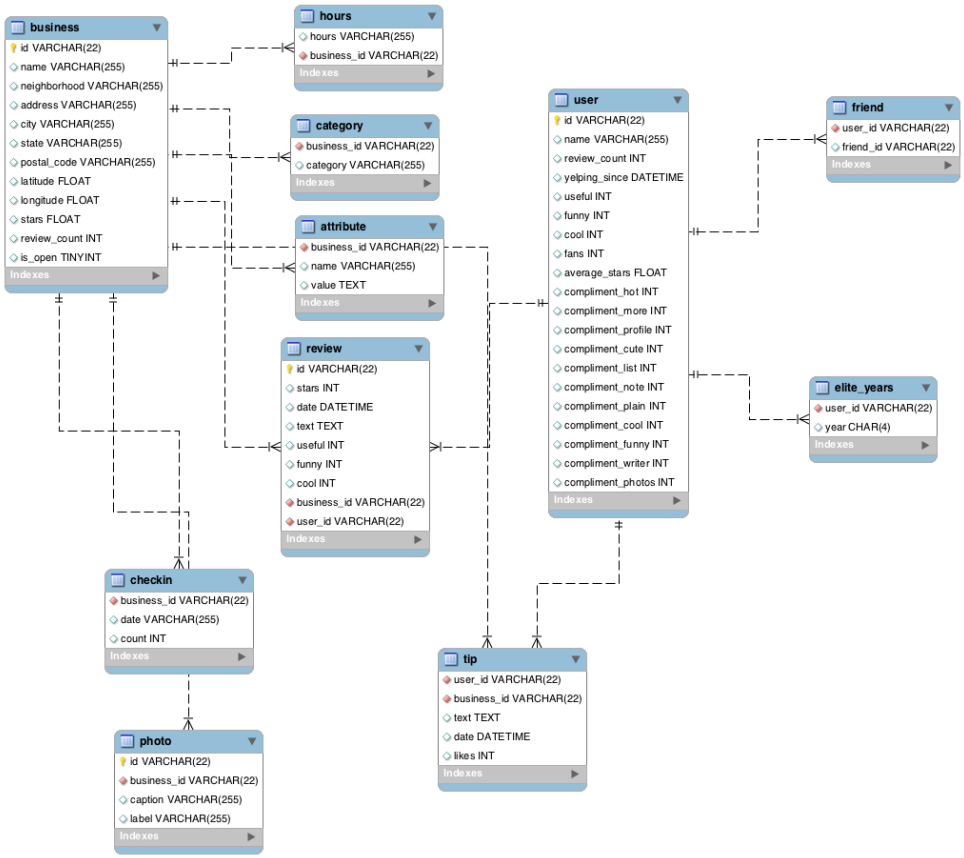
Challenges

There's a few different challenges we faced when working with the Yelp dataset:

1. The dataset is massive. There are ~5M reviews, and ~150K businesses. It's ~6 GB in size as well.
2. Given its massive size, processing it takes a ton of resources. I realized a bit too late that I could create a MongoDB instance / create a server and use SQL to query subsets of the data instead, which I'll do in the future. Running all of the SVM models took a ton of time.

Methodology

1. We studied the business and checkin JSON files. The relation between the attributes and the table relations are shown below: (Source: Yelp)



2. We further narrowed down our objective to classify star ratings and more clearly figure out the correlation between the different parameters. We narrowed to the Business.json and Checkin.json files.

    a. Business.json gives us the stars, state, country, and category for all the business_ids
    b. Checkin.json gives us the count of checkins for each business for each hour for a year. We joined both of these and then calculated the number of checkins for each businesses, and the final dataset had the features from both of these json files into one.
3. We then normalized the fields and created the set of features (See pre-processing)
4. We only filtered restaurant records to further narrow down our dataset for ease of processing.
5. We then divided into 75%-25% ratio for the train-test split, and then applied the different models and compared them across each other based on the accuracy, recall, precision, and runtime.

Technologies

- We used Python on Jupyter Notebooks, and various data science libraries associated with Python such as:
    - Scikit-learn – ML Algorithms
    - Pandas – Data Manipulation
    - Numpy – Larger arrays and matrices
    - Seaborn – Data viz
    - Matplotlib – 2D plotting

Data Pre-Processing

As stated above, we're focusing on business.json and checkin.json for the purpose of predicting star ratings. We joined these on business_id and filtered the following attributes from business.json: stars, review, count, city, state, categories and created checkin_count for each business_id from Checkin.json.

We used pair-plot from seaborn to understand any correlation between the features and show the result in the appendix. (Figure 2). The data clearly is not normalized and we also realized that both the City and State are string values so we had to encode these using LabelEncoder from sklearn's preprocessing package so we can use these.

Now, to normalize. To do this, we used the roundoff function from pandas DataFrame class which rounds everything to the nearest decimal points, giving us values from 1-5 for the possible star values, then applied min – max normalization on the float values of checkin_count and review_count. We then dropped the columns that we were not going to use, and use the fillna function to fill an NaN values. Then, as mentioned above, we filtered so we only have restaurants to further narrow our scope by 3x. Our data is now ready for us to analyze. We also created a heatmap to further explore the correlation between the features (Figure 3).

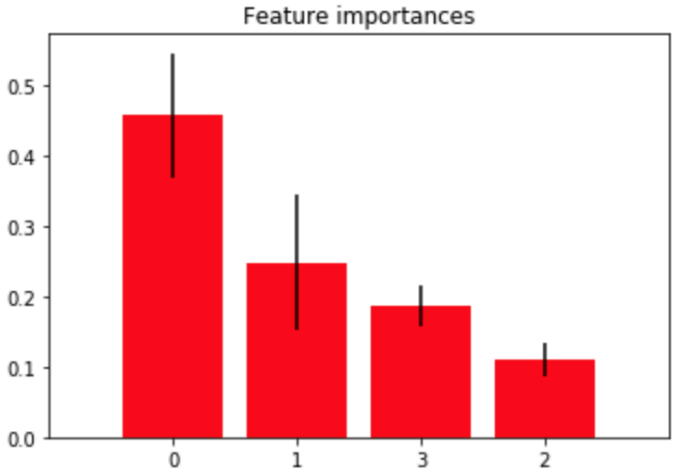
We discovered another issue – our feature set [review_count, checkin_count, state, city] with the label set as [stars] all had different feature with different ranges as well, so we used StandardScaler to normalize all of these as well.

After all the pre-processing, we split our data into training / testing sets to implement the different models which are below

Implementing the Models: We ran **5** different models and tuned their hyper parameters.

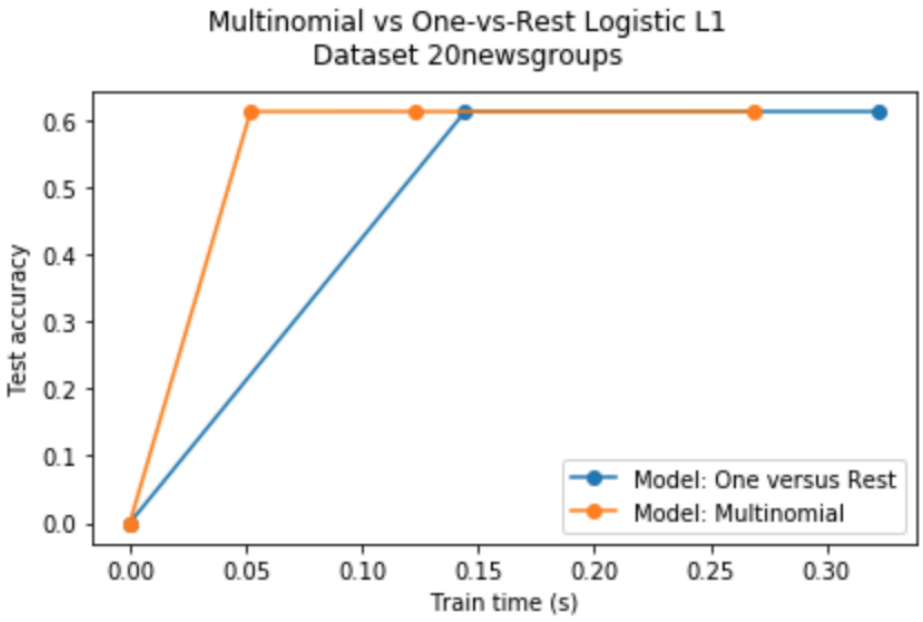
Random Forest

- Parameters:
  - n_estimators: Number of trees in the forest.
  - Max_depth: Maximum depth of the tree.
  - Random_state: Seed used by the random number generator,
- Interesting Observations:
  - Increasing the value of n_estimators causes the accuracy to increase, but beyond ~500, the accuracy goes down.
  - Runtime increases when increasing n_estimators as well.
- Accuracy Precision Score: [0, .0489, 0, .993, 0]
- Feature Ranking
  - Review_count: .456
  - Checkin_count: .248
  - State: .186
  - City: .110



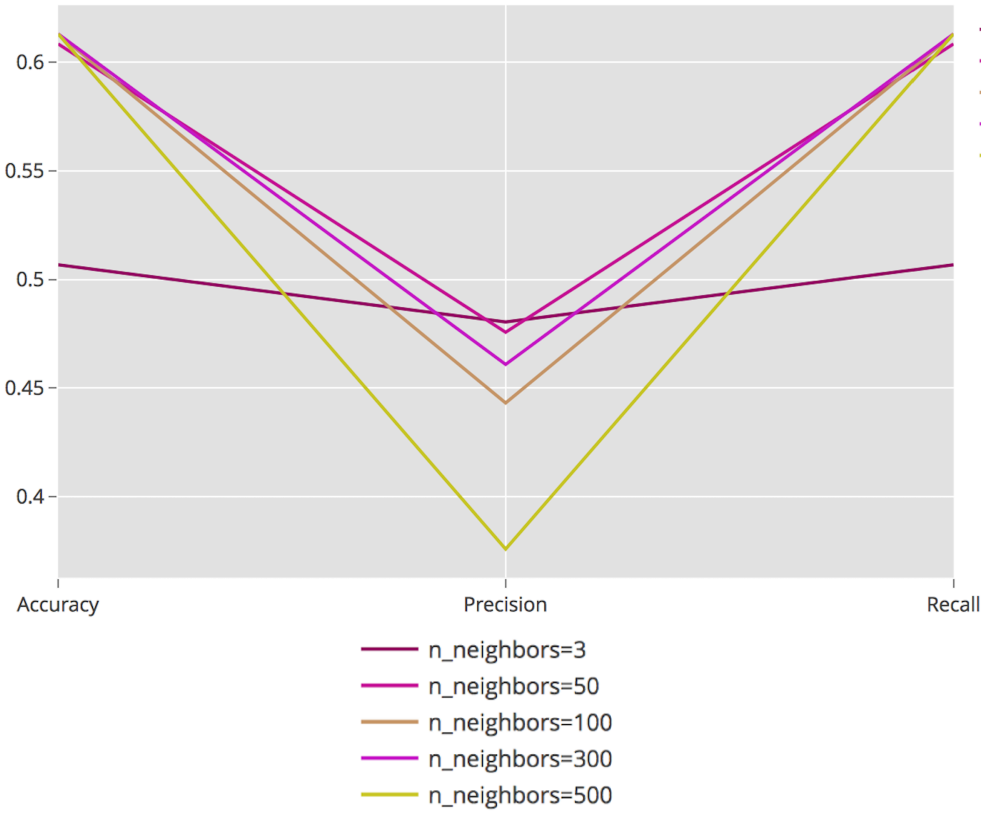
Logistic Regression

- Parameters:
  - Multi_class: Can be OVR or multinomial.
  - Tol: Tolerance for stopping
  - Solver: Algorithm to use to optimize
  - Max_iter: Max number of iterations for solver to converge to optimal value
- Observations
  - Changing Multi_class to OVR, decreasing tolerance, changing solver to newton-cg, lbgs, and saga, and changing max_iter from 20 to 2000 did not cause significant changes in accuracy.
  - Multinomial reaches 0.6 accuracy faster, and remains steady as well at around 0.6.



KNN

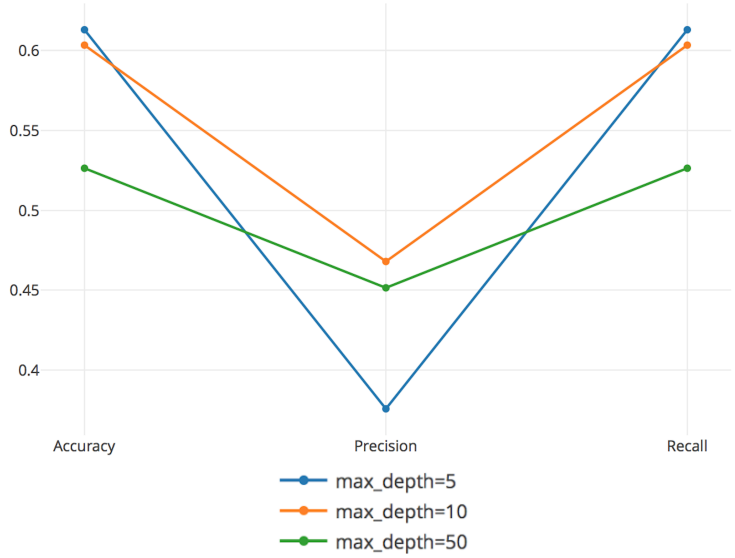
- Parameters:
  - n neighbors - number of neighbors to use for k-neighbors.
  - algorithm - algorithm to compute nearest neighbors.
  - weights - weight function used in prediction.
  - p - Power parameter for the Minkowski metric.

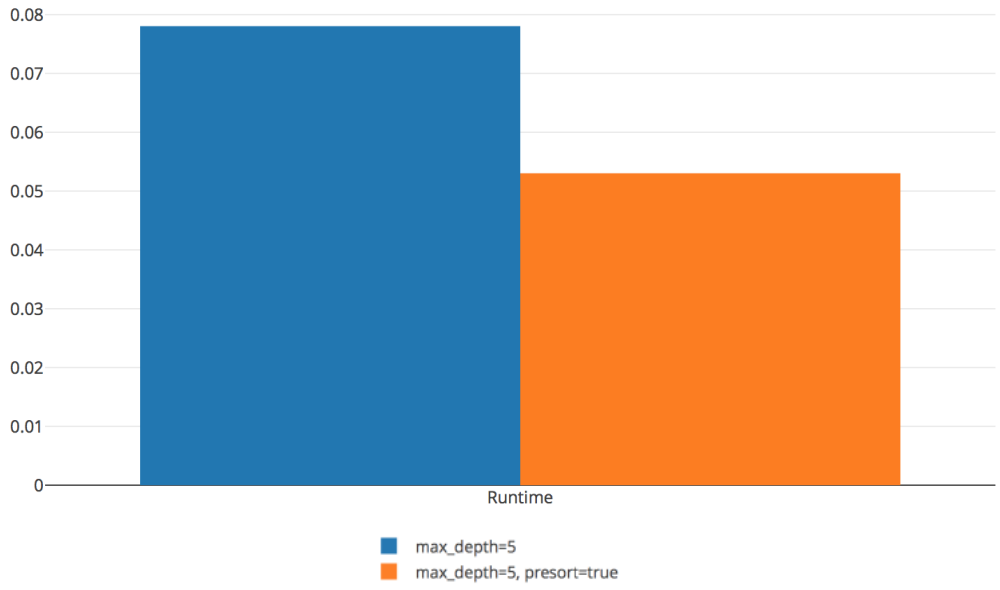


    - Looks like we're getting ~0.6 accuracy here as well.
- Observations
    - Increasing the value of n_neighbors declines accuracy when the value is beyond 100
    - If we play with the algorithm parameter while keeping n_neighbors constant, we get around ~62%, but this takes 10x the time to compute.
    - Setting the power to 5 has the highest accuracy at around 62%.

Decision Trees

- Parameters
    - Max_depth: Specifies the max depth of the tree
    - Presort: Specifies whether to presort the data to speed up findings of the best splits in fitting.





- Observations
    - Default parameters have a lower accuracy compared to when specifying max_depth (also a higher runtime)
    - Increasing max_depth from 5 all the way to 100 causes a steady growth in accuracy, but declines beyond 100.
    - Specifying presort (see above) causes a hefty difference.

SVM

- Parameters
    - Kernel: Specifies the kernel to be used in the algorithm
    - Gamma: The kernel coefficient
    - C: States the penalty parameter of the error term.
- Observations
    - RBFSVC performs better in accuracy and precision, but it takes 2x the time to run.
    - Keeping gamma constant while changing C to 1 increases accuracy, but slightly. Also a 2x increase in runtime.
    - Highest accuracy achieved with SVM with gamma set to 1000.

Concluding Remarks

SVM served as the best model for Yelp data, since it has the highest accuracy (~88%), highest precision (~88%), and highest recall (~88%).

Initially what we wanted to do with this project was to crunch all of the Yelp data and create a system which analyzed their review data and is able to tell them what changes they should make to their services. An increase in star rating by one star leads to an increase of ~8-10% in revenue, which is a sizeable difference. However, having lack of engineering man power caused us to pivot over to create more of a technical paper / create some sort of pipeline that we can grow on later to create this system.

Being able to show that features such as the number of reviews, number of checkins, state, and city correlate to having an increased number of stars could make a difference for business owners perhaps when they might want to expand, figure out where they'd like to open a new restaurant, etc.

I'd like to thank Ken Singer for facilitating a great course which really had myself hustling for an entire semester and Bailey Farren for being an excellent point person and coordinator. I'd also like to thank Shomit Ghose for his guidance and insight throughout the weeks.

Appendix / Graphs

Figure 2: Pairplot

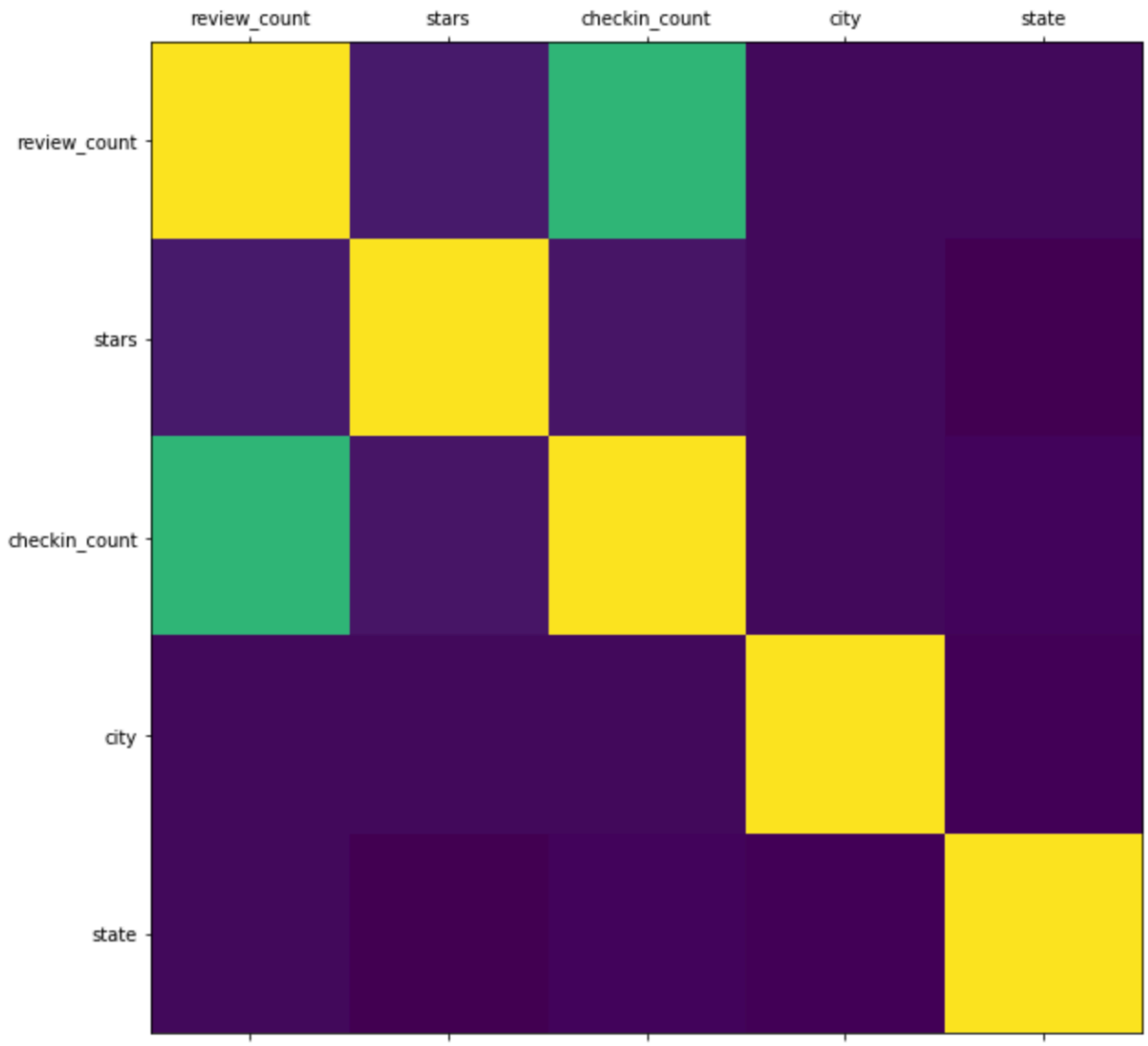


Figure 3: Heatmap showing correlation of features