Multi-label Classification Algorithms On Yelp Dataset

Raghav Nyati
The Department of Computer Science
San Diego State University
California, USA
Email: raghavnyati90@gmail.com

Kimaya Desai
The Department of Computer Science
San Diego State University
California, USA
Email: kimaya.desai@gmail.com

***Abstract*—The report explores the trend of various parameters in seven multi-class classification models. The objective of the experiment is to understand the impacts of these parameters on the model and to find the best model for each algorithm for the YELP Dataset. It analyzes different models in terms of accuracy, precision, recall-rate and run time.**

I. Introduction

The parameters of a Machine Learning model directly impact its performance. It is crucial to understand the trend of these parameters as this will help us derive a model that does a good job at predicting for new data. We use the YELP dataset to explore this trend.

The YELP dataset contains reviews of businesses across 12 metropolitan areas. Ratings & review directly influence the revenue and result in more visibility of the business. Majority of consumers incorporate star ratings to judge a businesses. So we use relevant attributes from the dataset to train the model to find out the star rating of a particular business. This is a multi class classification problem as our label will be one among the five stars.

A. Challenges

The various challenges that we realized with the YELP dataset are:

1) The YELP dataset is huge. It contains data of around 5 million reviews, 156,000 businesses. The dataset consists of 5 JSON files with over 1.2 million business attributes. The size of raw data 5.5GB.
2) The biggest drawbacks of the huge dataset is time taken for processing and taking in account irrelevant information and thereby wasting the resources.
3) It is tricky to find the correlation between these attributes and analyzing the contributing factors.

B. Solution

We identify the attributes that are relevant to the label we want to predict by finding the correlation between them. This narrows down the focus and eliminates the unnecessary information. Along with that grouping of relatable data into a file is required as the data spans across multiple JSON files. Further data needs to be cleaned. All these steps will utilize less resources also reducing the time taken. Considering the fact that YELP dataset is humongous, a rational approach is inevitable. Firstly, we studied the parameters in six JSON files. The attributes and table relations can be attributed as shown in the Fig 1. Secondly, we defined our objective to classify the restaurants stars rating on the viable parameters and to clearly understand the viable parameters we identified the correlation between the different parameters as explained in Data Preprocessing section below and this way we narrowed our search to two JSON files Business.json and Checkin.json. Business file gives us the stars, state, country and category for our all business ids. While Checkin file gives the count of total checkins happened on the specific business in every hour for a year long data. We combined both the files and using a python script calculated the total number of checkins for every business and in the final dataset combined the features from two JSON files into one.

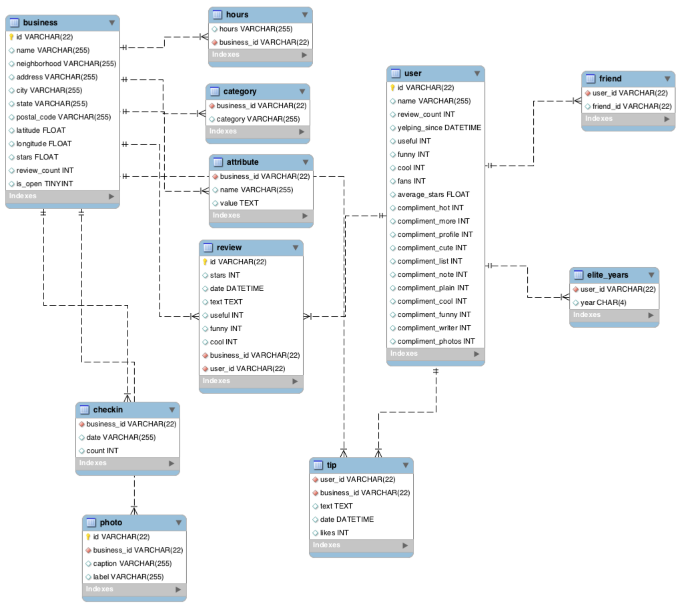


Fig. 1. Yelp Dataset : Association and structure of the tables.

The new files has 135147 records with the stars visibility as apparently shown in Fig 2.

Now, we have our dataset ready nonetheless not normalized.

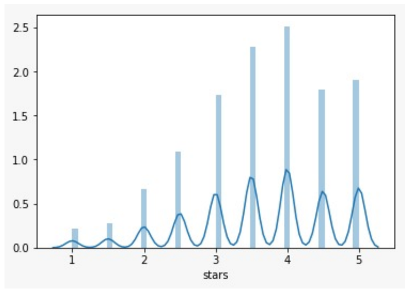


Fig. 2. Distribution plot of stars

We normalized the fields and created the feature set. The process is explained in detail in the data preprocessing step below. On next step, we filtered only restaurants records to ease the processing of total data and see the impact. This further narrowed the dataset records to 49403 records.

And finally we divided the dataset into 75%-25% ratio for training-testing dataset and applied the different variants of eclectic models for multi-class classifications and compared the best versions from all models in the final conclusion on accuracy, precision, recall-rate and runtime for the best variant of varied models.

II. Initial Set Up

Python offers flexibility with easy implementation and minimal syntax. We use the machine learning development environment of Python on Jupyter-Notebook. Python also offers helpful libraries such as sklearn which is a part of scikit-learn for Machine learning algorithms and implementations, pandas for data structures and analysis, numpy for large and multi dimensional arrays and matrices, seaborn for statistical data visualization, matplotlib for 2D plotting to produce publication quality figures which we will be required in the project. The installation of all these libraries is done using the pip command in Jupyter-Notebook python kernel. The easy installation can be done in Jupyter notebook python environment using below command after importing pip:

pip.main([”install”,”numpy”])

III. Data Preprocessing

The attributes and labels that we are interested in, for the prediction of star rating span across two files - business.json and checkin.json. We merged the two JSON files on the business_id and filtered the 5 attributes (stars, review_count, city, state and categories) from Business.json and one newly defined attribute named as checkin_count for every business_id from Checkin.json. To understand the correlation between the different fields we used pairplot from seaborn library and got the result as shown in below figure:

At this point the data is not normalized. On scrutiny, we realized that city and state are String values. Therefore, we applied LabelEncoder from sklearn.preprocessing package on state and city columns to identify all unique strings with a unique integer value.

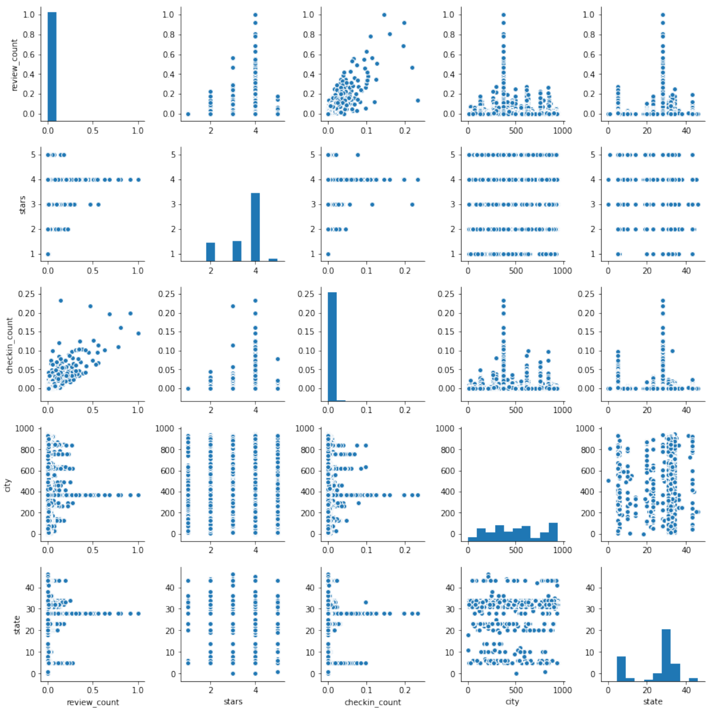


Fig. 3. Pair Plot depicting correlation of features

To normalize stars, we applied the round-off function from pandas dataframe which round every value to the nearest decimal point which made the possible values for stars as [1.0, 2.0, 3.0, 4.0, 5.0].

We applied the min-max normalization on the float values of checkin_count and review_count columns. And finally we dropped all the non-required columns. At this point we also check for null values if present in the column we applied fill function to manipulate the null field. Moreover to further ease the process, we filtered the data only category as Restaurants which narrowed our data from 135147 to 49403 records. At this point we believed our feature set ready. Nevertheless, to understand the feature correlation in depth we tried different ways of plotting the correlation including histogram, density plot, scatter matrix and heatmap as in Fig 4:

At this juncture, we made our feature set as [review_count, checkin_count, state, city] and label set as [stars]. Since this is a supervised learning we wanted to know total records available for different stars rating:

Star	Quantity
4.0	30596
3.0	9055
2.0	8331
5.0	1186
1.0	271

Moreover, here we realized that our feature set has different features with values in different ranges and to standardize the features on one scale we used StandardScaler to further

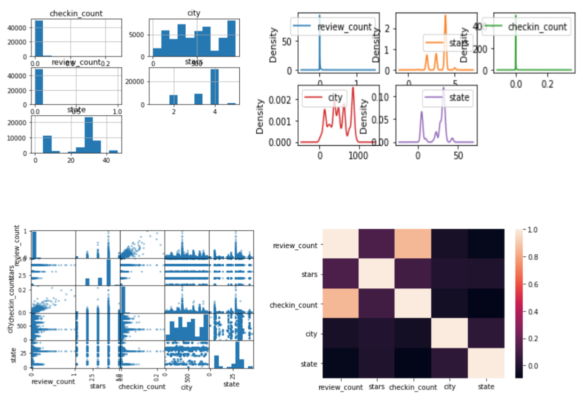


Fig. 4. (a) Histogram depicting correlation of features (b) Density-plot depicting correlation of features (c) Scatter-Matrix depicting correlation of features (d) Heat-Map depicting correlation of features.

transform the feature set. Post this we splitted the data into 4 parts features_train, label_train, features_test and label_test set. And we kept the ratio of training and testing data as 70%-30%. This prepared us to implement the different models.

IV. Model Implementation

The data is divided for training and testing. We run the data on 7 models by tuning their respective parameters.

A. Random Forest

1) Training Parameters:

- n_estimators : Number of trees in the forest.
- max_depth : The maximum depth of the tree.
- random_state :The seed used by random number generator.

2) *Testing Model:* Random Forest

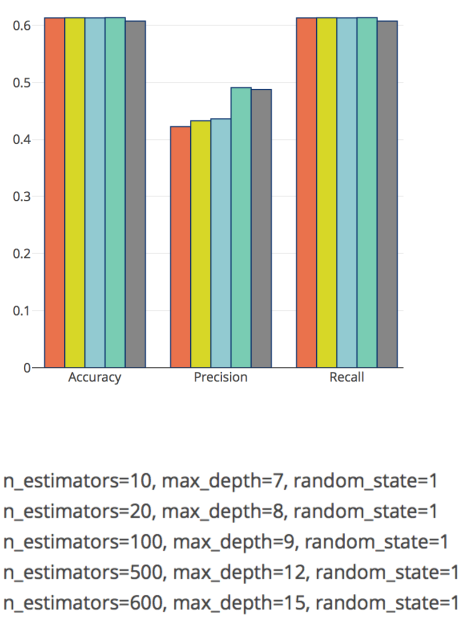


Fig. 5. Random Forest Analysis: Increasing n_estimators and max_depth

3) *Observations::*

- On keeping max_depth constant, when the value of n_estimators is low, accuracy achieved is good. If we increase value of n_estimators upto 500, accuracy increases. However, beyond 500 the accuracy starts to decline.
- On keeping n_estimators constant,and increasing max_depth, the accuracy increases. But when the max_depth is kept beyond 8, the accuracy starts to decline.
- On increasing n_estimators, runtime increases proportionally.
- The best of random forest model gave the following observations:
  - Confusion Matrix:

$$\begin{array}{ccccc} 0 & 2 & 0 & 70 & 0 \\ 0 & 101 & 0 & 1999 & 0 \\ 0 & 31 & 0 & 2268 & 0 \\ 0 & 56 & 0 & 7516 & 0 \\ 0 & 2 & 0 & 306 & 0 \end{array}$$

  - Class Precision Score:
    [0, 0.52604167, 0, 0.61814294, 0]
  - Accuracy Precision Score:
    [0, 0.04809524, 0, 0.99260433, 0]
    - Feature ranking:
      feature 0 (0.456283) - 'review_count'
      * feature 1 (0.248002) - 'checkin_count'
      * feature 3 (0.185962) - 'state'
      * feature 2 (0.109753) - 'city'

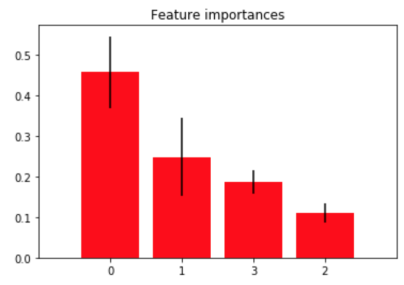


Fig. 6. Feature ranking depicting feature importance.

B. Logistic Regression

1) Training Parameters:

- multi_class - Multi-class option can be either ovr or multinomial. If the option chosen is ovr, then a binary problem is fit for each label. Else the loss minimised is the multinomial loss fit across the entire probability distribution.
- tol : Tolerance for stopping criteria
- solver : Algorithm to be used for optimization
- max_iter: Maximum number of iterations taken for the solvers to converge.

2) Testing Model: Logistic Regression

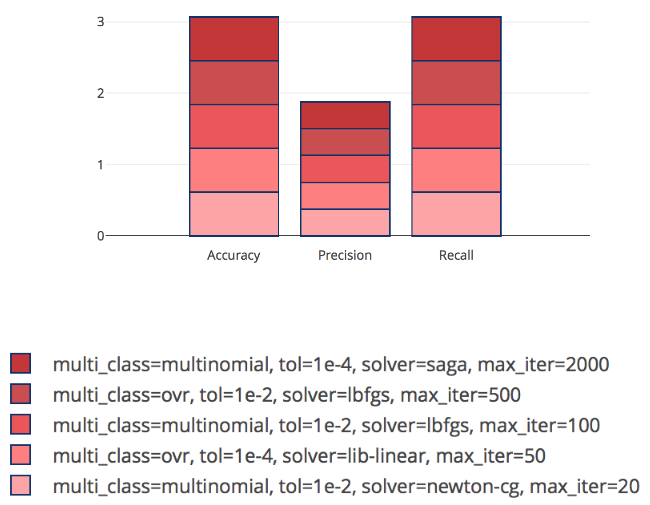


Fig. 7. Logistic Regression Analysis

3) Observations::

- On changing the following parameters:
  multi_class as ovr and multinomial
  tol from 1e-1 to 1e-5
  solver as newton-cg, saga, lbgs
  penalty as L1, L2
  max_iterations from 20 to 2000
  There is no change in accuracy, precision or recall. Also, we observed no significant trend in runtime.
- On plotting graph for multi_class parameter, we compare the accuracy of ovr and multinomial where in we observe that the test accuracy increases over train time, Multinomial reaches 0.6 accuracy faster and at 0.6, the accuracy remains steady for both multinomial and ovr.

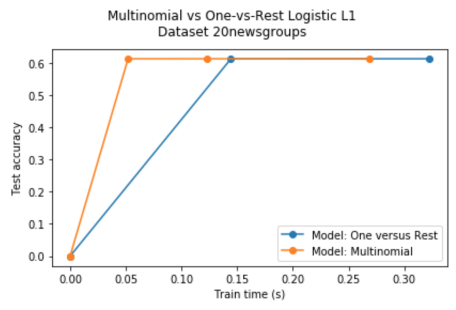


Fig. 8. Logistic Regression: Multinomial vs One-vs-Rest Logistic L1

C. kNN

1) Training Parameters:

- n_neighbors - number of neighbors to use for kneighbors queries.
- algorithm - algorithm to compute nearest neighbors.
- weights - weight function used in prediction.
- p - Power parameter for Minkowski metric.

2) Testing Model: K Nearest Neighbor

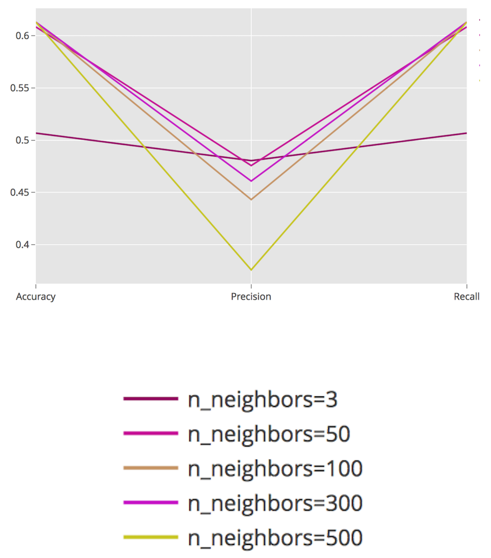


Fig. 9. K Nearest Neighbor Analysis : Increasing n_neighbors

3) Observations:

- On default parameters, an accuracy of 53.97% is achieved. However, on inclusion of n_neighbors as 3 the accuracy drops.
- When the value of n_neighbors is increased the accuracy increases slightly until 100. However, it starts to decline on increase in neighbors beyond 100.
- On keeping n_neighbors constant and adding the algorithm parameter - if we use the algorithm as ball-tree, the accuracy achieved is 61.67% which is slightly better than the accuracy achieved on using brute algorithm. However the time-taken by brute algorithm is 14 times the time required by ball-tree.
- The weights when set to distance, the accuracy decreases, whereas when the weights are set to uniform the accuracy increases. This is because in distance the closer weights have more importance which in turn reduces the accuracy.
- Power Parameter when set to 5 shows the highest accuracy at 61.69%. When it is reduced to 2 or 1, it shows slight difference.

D. MLP Classifier

1) Training Parameters:

- hidden_layer_size : Represents no of neurons in the hidden layer.
- activation_function : Defines the activation function for the hidden layer.
- alpha: L2 penalty (regularization term) parameter.
- solver : For weight optimization.
- learning_rate: For weight updates.
- max_iter: Maximum no of iterations.

2) Testing Model: MLP Classifier

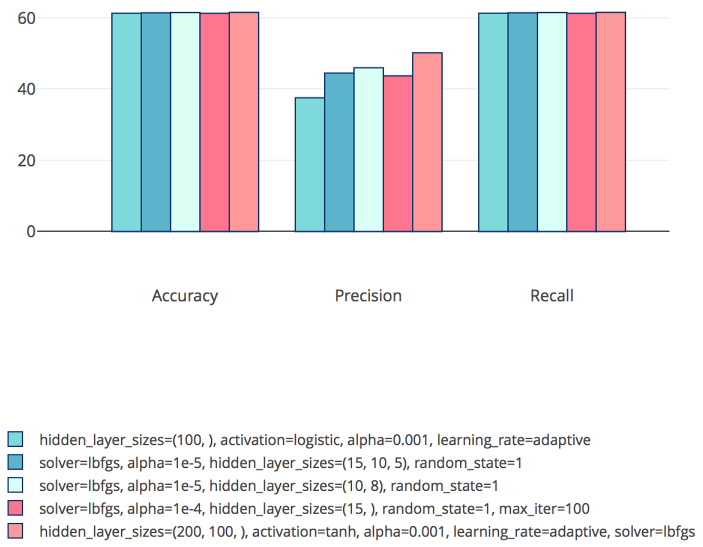


Fig. 10. MLP Analysis

3) Observations:

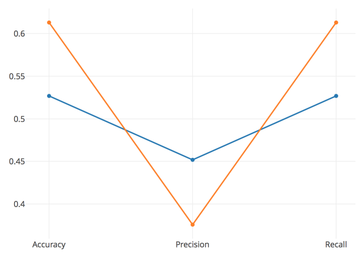
- Using default parameter vs adding hidden_layer_sizes, there is hardly any change in the accuracy. However, the runtime increases exponentially on addition of more hidden layers and parameters.
- On changing activation_function to tanh from logistic, the accuracy increases.
- On adding alpha to be 1e - 5 and solver as lbfgs, we get a higher accuracy. However, on increasing the hidden layer sizes on above computation, the accuracy shows a slight drop.
- Also, on adding max_iter as 100, the accuracy shows a further drop.
- Precision is higher on introducing the parameter activation_function as tanh. On further addition of learning rate as adaptive, it increases further.

E. Decision Tree

1) Training Parameters::

- max_depth : Specifies the maximum depth of the tree.
- presort : Specifies whether to presort the data to speed up the findings of the best splits in the fitting.

2) Testing Model: Decision Tree



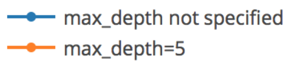


Fig. 11. Decision Tree: Default parameter vs max_depth = 5

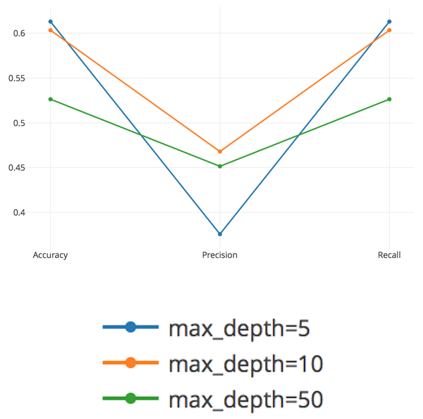


Fig. 12. Decision Tree: Increasing max_depth

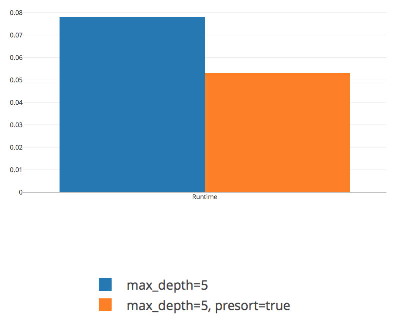


Fig. 13. Decision Tree : Specifying presort as true.

3) Observations:

- When default parameters are specified in decision tree, accuracy is low as compared to when we specify max_depth as a parameter. Also, the runtime taken is higher.
- On increasing max_depth from 5 up to 100, the accuracy shows a steady growth. However, beyond 100, the accuracy shows a decline.
- On specifying presort as the parameter, along with max_depth constant, the runtime shows a steep decrease.

F. SVM

1) Training Parameters:

- Kernel : Specifies the kernel type to be used in the algorithm.
- Gamma : Defines the kernel coefficient.
- C : States the penalty parameter of the error term.

2) Training Time:

RBF vs Kernel

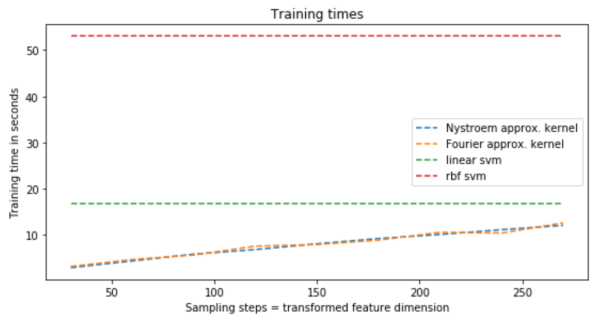


Fig. 14. Training Time - RBF vs Linear

3) *Testing Model:* .

SVM

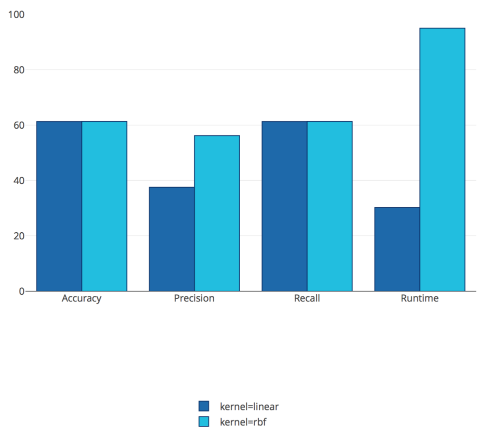


Fig. 15. SVM Analysis: Kernel Comparison

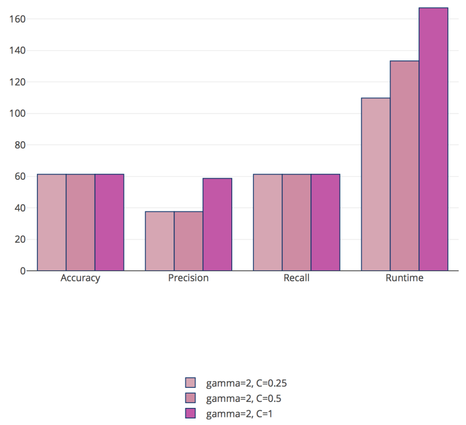


Fig. 16. SVM Analysis: Increasing C

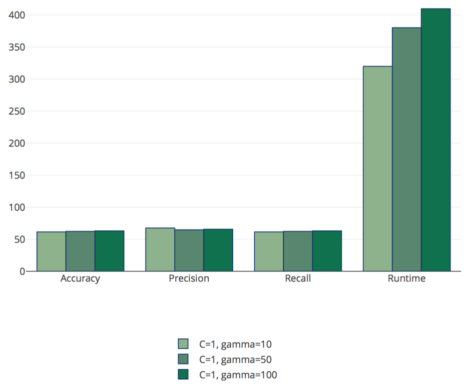


Fig. 17. SVM Analysis: Increasing gamma

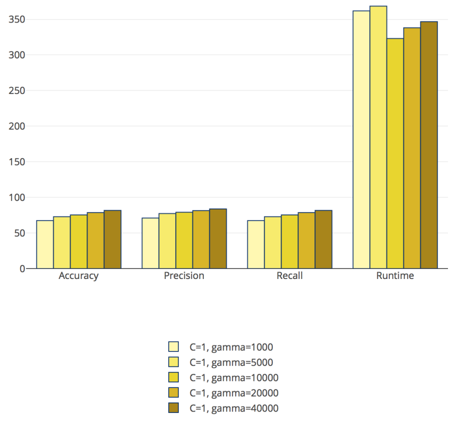


Fig. 18. SVM Analysis: Increasing gamma

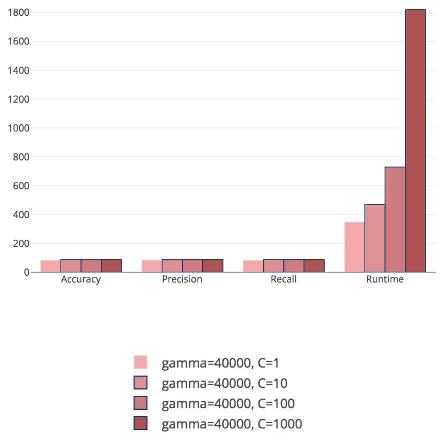


Fig. 19. SVM Analysis: Increasing gamma

4) *Observations:*

- On changing the kernel from LinearSVC to RbfSVC, RBF performs better in terms of accuracy and precision. However it takes the twice amount of time to run RBF as compared with LinearSVC.
- On keeping gamma constant, a change in value of C from 0.025 to 1 increases the accuracy slightly. Also, there is a significant increase in run time.
- On keeping C constant, a change in gamma increases the accuracy However, it decreases precision.
- The highest accuracy so far of 65.61% was achieved at

gamma set as 1000.

G. Naive Bayes

1) Bernoulli Naive Bayes: When default parameters are set for Bernoulli, we achieve an accuracy of 61.3%, precision of 37.5% and recall rate of 61.3%. The runtime for Bernoulli Naive Bayes achieved is 0.045 secs.

2) Gaussian Naive Bayes: When default parameters are set for Gaussian, we achieve an accuracy of 21.5%, precision of 55.6% and recall rate of 21.5%. The runtime for Gaussian Naive Bayes achieved is 0.036 sec.

V. Results & Conclusion

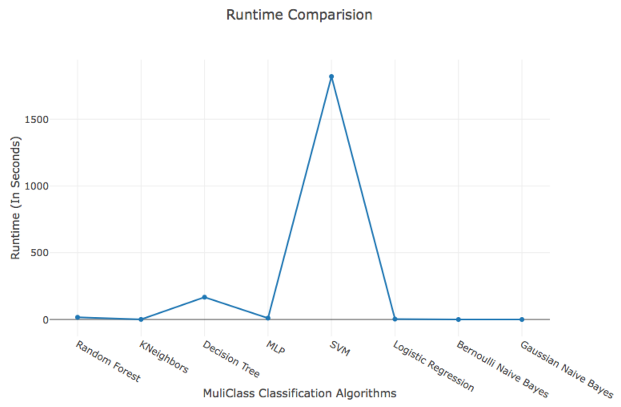


Fig. 20. Recall

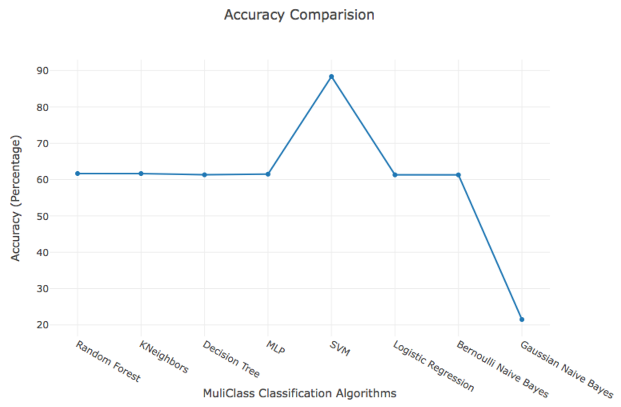


Fig. 21. Accuracy

Best Algorithms in terms of:

- Runtime - Gaussian Naive Bayes 0.036 secs
- Accuracy - Support Vector Machine 88.37%
- Precision - Support Vector Machine 88.59%
- Recall - Support Vector Machine 88.37%

Therefore, the Support Vector Machine Model suits best for the YELP Dataset as it has the highest accuracy, precision and recall. Gaussian Naive Bayes has the highest runtime, however

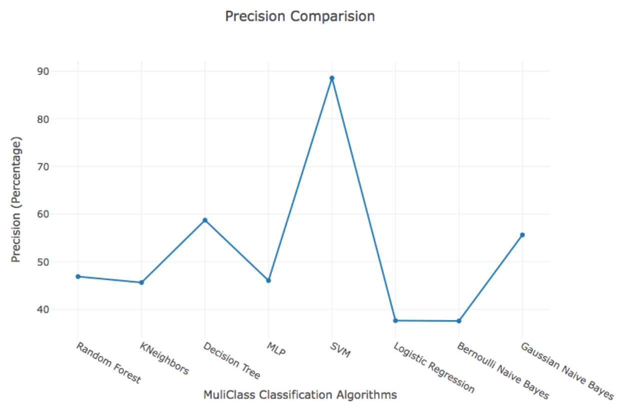


Fig. 22. Precision

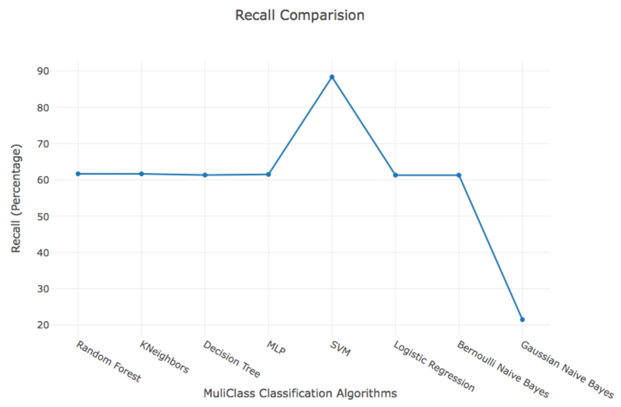


Fig. 23. Recall

the accuracy obtained is merely 21.5% which is the lowest of all.

References

[1] Buitinck, Louppe et al, *API design for machine learning software: experiences from the scikit-learn project*, ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013.

[2] Yelp Dataset Challenge. Accessed (2017, December 09). Retrieved from https://www.yelp.com/dataset/documentation/json

[3] Pandas Documentation. Accessed (2017, November 27). Retrieved from https://pandas.pydata.org/pandas-docs/stable/install.html