Project Report

Predicting ratings for reviews stored in NoSQL datastore

Instructor	Prof Gheorghi Guzun
Course	CMPE 297 Big Data Algorithms
Group No	3

Team Information

Team Members	SJSU ID	Email ID
Saranya Visvanathan	014483625	saranya.visvanathan@sjsu.edu
Manish Arigala	014492712	manish.arigala@sjsu.edu
Sakshi Ahuja	015266823	sakshi.ahuja@sjsu.edu

Project Description

A lot of e-commerce companies rely on reviews and ratings provided by the customers to build their business models. The product recommendations given to the customers also heavily rely on the previous ratings and reviews provided by the customers. Hence ratings and reviews play an important role in driving the business forward. However, there are some cases where a review is disproportionate to the rating given, which could disrupt the business model. So, this project focuses on predicting the ratings from yelp reviews using NLP techniques. A detailed description of the implementation, requirements, and challenges are discussed in this report.

Dataset description

DB Schema

Database	yelp
Collections	reviewsTrain, reviewsTest
collections Schema	**Keys:** id, stars, reviews

Table 1

```
MongoDB Enterprise atlas-9psnpx-shard-0:PRIMARY> show dbs
admin  0.000GB
local  2.385GB
yelp   0.343GB
MongoDB Enterprise atlas-9psnpx-shard-0:PRIMARY> use yelp
switched to db yelp
MongoDB Enterprise atlas-9psnpx-shard-0:PRIMARY> show collections
reviewsTest
reviewsTrain
MongoDB Enterprise atlas-9psnpx-shard-0:PRIMARY> db.reviewsTrain.findOne()
{
        "_id" : ObjectId("5fadf7f73b741b26279c9fbd"),
        "stars" : "1.0",
        "review" : "Dismal, lukewarm, defrosted-tasting \"TexMex\" glop;\n\nMumbly, unengaged waiter;\n\nClueless manage
r, who seeing us with barely nibbled entrees\non plates shoved forward for pickup, thanked us\nperfunctorily for our pat
ronage;\n\nWe're from the Texas Hill Country;\ndown there, we jail critters \nwho serve up grub this bad,\nfor their own
 protection.\n\nNever, never, NEVER again\n(Back to Yard House for real food)"
}
MongoDB Enterprise atlas-9psnpx-shard-0:PRIMARY> db.reviewsTest.findOne()
{
        "_id" : ObjectId("5fadfaaf24a5449abd258ed9"),
        "stars" : "5.0",
        "review" : "I went out with a bunch of my friends and had an amazing time at divas. Such a fun show, and such a
professional cast. The dancers were on it and very together! Th queens were true to character and blew us away. Frank ha
d us laughing the entire show as well. I would advise anyone and everyone to come to this show!!"
}
MongoDB Enterprise atlas-9psnpx-shard-0:PRIMARY>
```

Fig. 1

Data statistics

Yelp data is processed and created as balanced data. Dataset preparation is explained in detail here. The following image shows the training data distribution.

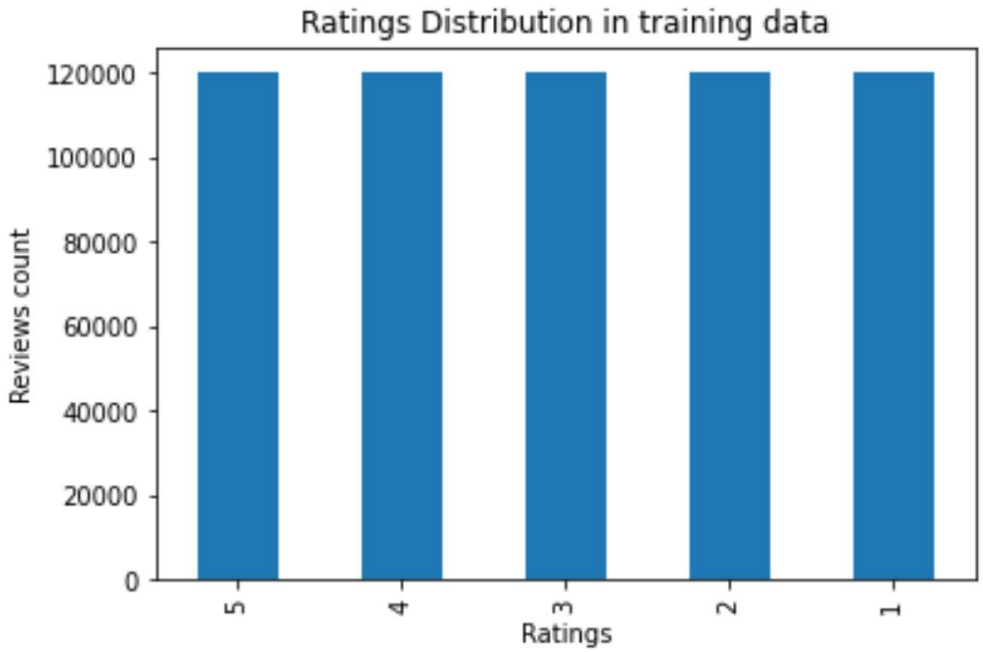


Fig. 2

Project architecture

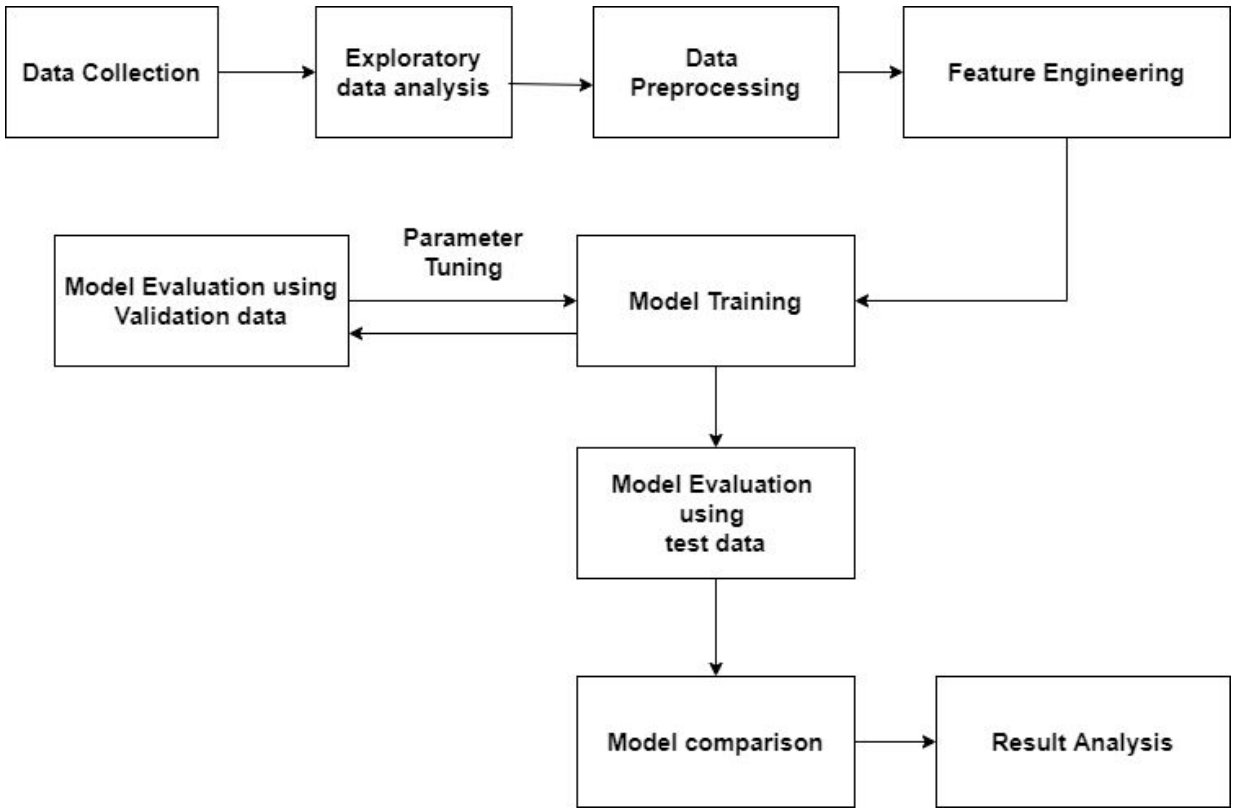


Fig. 3

Project Implementation

Dataset Preparation:

Machine learning algorithms learn from data. It is critical that you feed them the right data for the problem you want to solve. This step is concerned with selecting the subset of available data to train the models. Initially, there were approximately 8GB of data. And, each document looked as below.

root":{*9 items*

```
  "review_id":string"xQY8N_XvtGbearJ5X4QryQ"
  "user_id":string"OwjRMXRC0KyPrIlcjaXeFQ"
  "business_id":string"-MhfebM0QIsKt87iDN-FNw"
  "stars":int2
  "useful":int5
  "funny":int0
  "cool":int0
"text":string"As someone who has worked with many museums, I was eager to visit this gallery on my most recent trip to Las Vegas. When I saw they would be showing infamous eggs of the House of Faberge from the Virginia Museum of Fine Arts (VMFA), I knew I had to go!"
  "date":string"2015-04-15 05:21:16"
}
```


Fig. 4

There were approximately 8 million documents in total. Considering the limit of 512 MB of MongoDB Atlas, approximately 1 million records are taken and created as collections. Stars and text are mapped to stars and review respectively.

After the processing, each document has a schema as shown below.


```
{
        "_id" : ObjectId("5f9e5249a485dc43db4ce4b8"),
        "stars" : "5.0",
        "review" : "This is definitely my favorite fast food sub shop. Ingredients are everything, and everything I see and taste here tells me that they're using top-grade fresh ingredients. The brisket sandwich is probably my favorite... and it's the one my wife ALWAYS gets. Unlike her, I often bounce around the menu to try different things. Definitely a step up from Subway, Quiznos, Jimmy Johns, etc in my opinion. As with all of my reviews, I grade each place relative to what I perceive to be its peers - so five star compared to them."
}
```


Fig. 5

The collection is divided into two parts: training and testing data. After the dataset preparation, training data has a total of 599995 documents and test data has 149995 documents.

Data preprocessing - cleaning

The process of cleaning data is a very crucial step in Machine Learning and NLP. Cleaning of data helps in saving processing time and resources and getting better and more accurate results. Therefore, we followed the steps listed below to make sure our data is as clean as possible.

1. **Converting to Lowercase**
   Initially, we convert the entire text to lowercase

2. **Removing Punctuations**
   Punctuation marks don't usually add any value or meaning while training the models. Hence it becomes very important to remove these from the text. We used the string library's punctuations (32 punctuations) to remove them.

3. **Tokenization**
   Tokenizations is the process of converting a string to a List of Words. For this, we used Regular expressions to create the List of Words

4. **Removing Stop Words**
   Stop words are the most commonly used words, which don't help in training our models, hence these words must be removed. There is no universal standard for these words. Each library has its own set of stop words. We used the popular 'nltk' library to remove these stop words. Some of the words from this library are ['i', 'me', 'myself', 'we', 'our', 'ourselves', 'you', 'you're', 'you've']. There are a total of 179 stop words from this library.

5. **Removing Numbers, URLs, and Tags**
   We also removed numbers, URLs, and HTML tags from the dataset

6. **Lemmatization**
   A lemmatizer reduces a word to its root form. It considers the context of the word and converts it into its root form based on the dictionary's definition. For example, the root word 'belief' has many forms such as 'believing', 'believed', or 'believes'. Lemmatizer further reduces the size of the dataset consisting only of the root words. Hence, improving the speed and accuracy of training ML models.

7. **Remove repeated characters in words like "Wowwwww"**
   Removing repeated words makes the words meaningful. Sometimes these words contain human emotions like excitement/anger. Considering these words can be proved to be helpful in predicting the ratings.

8. **Keep only meaningful English words**
   Meaningless words are like noise. It is important that we keep significant words that are meaningful for adding accuracy while predicting the ratings.

Exploratory Data Analysis

Exploratory Data Analysis or EDA is an approach of analyzing data sets to summarize their main characteristics. In our project, we take the cleaned data to explore its main characteristics and features. Data distribution and dominating words in the data are analyzed.

We initially tokenize the data to get the Bag of Words. The total number of tokens was found to be 34403534.

We further filter out stop words from another library just to make sure we don't carry on some of the stop words. This further reduced the number of tokens leaving us with 31944921 tokens.

After getting the perfectly cleaned data, we used the nltk's library to plot the most frequently used words from all the reviews. The visualized plot is shown below

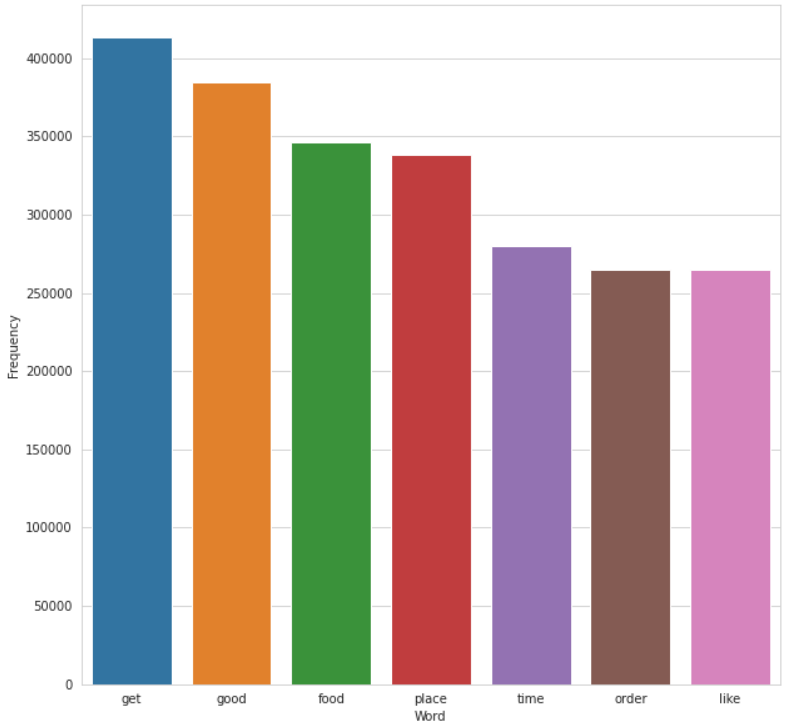


Fig. 6

As we can observe from the above figure, 'get', 'good', and 'food' were the top most used words from the entire yelp train dataset.

Index creation and performance analysis

Indexing is used to facilitate data access which helps to increase the execution speed. In this project, the TF-IDF vectorizer is used to create vectors for a set of documents. The analysis is done on the vectorization creation with and without providing an index as an input.

The inverted index is a data structure that maps individual terms to a set of values which can be either the document number(s) that it is located in or the position it occurs. There are many variants of the inverted index which are used as per the need. Creating tfidf vectors can be made faster by providing the pre-built dictionary of terms mappings.

One variant of the inverted index is creating a dictionary where keys are terms and values are the list of document numbers where it occurs. This variation is implemented using the HashedIndex package. Each word has been iterated and its location(s) are added to the dictionary. But, sklearn's TfidfVectorizer library has not provided any option to input this variant of the index. So, the parameters of the vectorizer package are analyzed to understand its expectations. It is found that it expects a dictionary where keys are arranged as it appears in the document and its values are integer numbers assigned in lexicographic order of the words stored as keys.

For example,

Sample document	["apple fruit","she ate apple"]
TfidfVectorizer Vocabulary's parameter expectation	{ "apple":0, "fruit":2, "she":3, "ate":1} [keys are ordered as they appear in the document Values are integer number assigned in lexicographic order of the words starting from index 0.]

Table 2

So, the above concept is implemented for creating indexes expected for tfidf sklearn package. But, our implementation is not very optimized which made the execution time of indexes creation quite longer (approx. for 30mins).

So, to analyze the indexing performance efficiently, the vocabulary created by tfidfvectorizer sklearn package is used as an input in the next run and the performance is analyzed. This is the analysis of the training data which has 600000 documents.

Technique	Total time	CPU time	System time
TfidfVectorizer without indexing	57.8s	57.1s	737ms
TfidfVectorizer with indexing	57s	56.6s	418ms

Table 3

There is definitely a difference in the performance. There is a 43.2% decrement in system execution time and approx. 1% decrement in CPU time. There isn't much reduction seen maybe because of the optimized implementation of TfidfVectorizer. Since this sklearn package has been in the market for a long time, a lot of improvisation has been done on top of it which is evident from its optimized performance without providing any pre-calculated information.

Train-test split

After the dataset preparation step, there are two different data available, one for training and one for testing. To evaluate the model training, training data is further split into train and validation data.

Vectorization

Vectorization is done to convert each review into a vector of numerical values that can be utilized with the model. A vector is created for each review using TfidfVectorizer for ML model training. So, Once the set of cleaned reviews is given as input, the CSR matrix is given as an output. The same is applied for training, validation, and test data.

The following snippet shows the size of tfidf vectors created for all the data.

```
#shape of tfidf vectors
print(f"training vectors shape: {xtrain_tfidf.shape}")
print(f"validation vectors shape: {xval_tfidf.shape}")
print(f"testing vectors shape: {xtest_tfidf.shape}")
```

```
training vectors shape: (479996, 300180)
validation vectors shape: (119999, 300180)
testing vectors shape: (149995, 300180)
```

Fig 7

So, for LSTM, the DL technique, GloVe embedding of 300 dimensions is used. A brief explanation of it is given here. Using glove, an embedding dictionary is created which is then used to create a vector of 300 dimensions for each review. Later, it is converted to an embedding matrix to give as an input to the DL model.

Dimensionality Reduction

The vectors created by tfidf have the dimension, 300180 which is quite high. So, the model took a long time (around 1-6 hours depending upon the models) to run even when trained with GPU in Google collab. So, the linear dimensionality reduction technique SVD (Singular Value Decomposition) is used to reduce the model dimensions.

TruncatedSVD - sklearn decomposition package

TruncatedSVD does not center the data which makes it work efficiently with sparse data. So, it is used to train the data created by tfidfvectorizer which is sparse. Also, in particular, TruncatedSVD works well with tfidf vectors. The below snippet shows the size of vectors after applying dimensionality reduction.

```
print(f"xtraindr shape: {xtraindr.shape}")
print(f"xvaldr shape: {xvaldr.shape}")
print(f"xtestdr shape: {xtestdr.shape}")
```

```
xtraindr shape: (479996, 100)
xvaldr shape: (119999, 100)
xtestdr shape: (149995, 100)
```

Fig 8

Building models

Regression models are trained to predict ratings given the reviews. The model is trained with training data and evaluated with validation data. Based on the errors of evaluation, the training model is tuned and trained again. Finally, the model with minimum evaluation error is used to predict the ratings on the test dataset.

For model training, sklearn packages are used. Model training is tried with both tfidf vectors (for some models) and dimensionality reduced vectors. It took a long time to train the model when tfidf vectors are used because of its dimensionality.

The following table lists the regression algorithms used for training.

Linear models	Logistic regression, SGD classifier, SVM
Tree	Decision Tree
Ensemble	Random Forest
Neural Network	MLPClassifier

Table 4

Hyperparameter Tuning

Hyperparameter tuning refers to adjusting the parameters passed to the models during training based on the feedback from the evaluation of the model on the validation data. The sklearn's package RandomizedSearchCV helps to tune the model parameters. Given a dictionary of possible parameters, it tries out different combinations and results in the parameters combination which provides better performance which would be decided by the error measure provided as an argument. Sometimes, the model predicted better with the default parameters itself.

The below snippet attached is the hyperparameter tuning for a random forest model regressor. There, the param_grid is a dictionary where the possible parameter values needed to be tried out can be given, among them, a set of parameters that gives the better performance is returned as the output.

```
[33]: n_folds = 10

      param_grid = {
          'max_depth': [80, 90, 100, 110],
          'max_features': [2,20,30,50,100,200],
          'max_leaf_nodes': [3, 4, 5, 8, 16, 32, 64],
          'min_samples_split': [0.1, 0.2, 0.3, 0.4, 0.5],
          'max_samples': [200, 500, 10000, 50000, 100000]
      }
      rfclf=RandomForestRegressor(random_state=42)
      grid =RandomizedSearchCV(rfclf,param_grid,scoring='neg_mean_squared_error',n_jobs=-1,cv=n_folds)

      result = grid.fit(xtraindr, ytrain)
      print(result.best_params_)

      {'min_samples_split': 0.1, 'max_samples': 50000, 'max_leaf_nodes': 16, 'max_features': 2, 'max_depth': 80}
```

Fig 9

Model Evaluation

The evaluation metric finalized is the Root Mean Squared Error (RMSE). RMSE calculates the standard deviation of the squared error of the predictions. Thereby, it figures out the variation between predicted and actual values, which helps to understand how close the target function is replicated by the trained model. The lesser the value of RMSE, the better the model performance.

The other evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAS), and r2 score are also considered. MSE and MAE calculate the mean of the squared error and absolute error of the predictions respectively, which helps to figure out the variation between prediction and actual value as similar to RMSE. r2 score identifies how well the data points are aligned to the regression line. So, the higher the value of the r2 score, the better the model performance is.

The regression algorithms are explained in detail here. The following includes the training results of these algorithms on the validation data in the project.

Logistic Regression

The random state is always set as 42.

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**

{solver: saga} **[Final hypothesis for Logistic Regression]**	0.850	0.723	0.492	0.639
truncatedSVD - 2 dimensions, standardscaler, {solver: saga}	1.624	2.639	1.136	-0.313
truncatedSVD - 100 components standardscaler, {solver: saga}	1.338	1.792	0.897	0.108
truncatedSVD - 100 components powertransformer, {solver: saga}	1.340	1.797	0.899	0.105

Table 5

Support Vector Machines(SGDClassifier)

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**
max_iter=1000, tol=1e-8,alpha=0.0001)	1.047	1.096	.641	0.454
max_iter=1000, tol=1e-3	1.035	1.072	0.630	0.466
max_iter=5000, tol=1e-3	1.041	1.084	0.635	0.460
max_iter=10, tol=1e-3,random_state=42,n_jobs=10 With tf idf	1.041	1.083	0.634	0.460
max_iter=1000, tol=1e-3,random_state=42 **[Final hypothesis for SGDClassifier]**	1.03	1.07	0.62	0.47

Table 6

LinearSVC

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**
random_state=0, tol=1e-05	1.258	1.584	0.840	0.211
penalty='l1', loss='squared_hinge', dual=False, tol=1e-3 **[Final hypothesis for LinearSVC]**	0.907	0.824	0.531	0.589

Table 7

Decision tree

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**
Default parameters (with dimensionality reduced vectors)	1.537	2.364	1.124	-0.176

With Tfidf vectors	1.296	1.681	0.891	0.163
{ min_samples_split: 10, min_samples_leaf: 5, max_features: 3, max_depth: 80 } (with dimensionality reduced vectors)	1.559	2.43	1.233	-0.209
{ min_samples_split: 10, min_samples_leaf: 5, max_features: 3, max_depth: 80 } (with tfidf vectors)	1.407	1.982	1.197	0.008
{min_samples_split: 0.3, min_samples_leaf: 64, max_features: 100, max_depth: 90} (with dimensionality reduced vectors)	1.226	1.504	1.004	0.251
{min_samples_split: 0.1, min_samples_leaf: 5, max_features: 100, max_depth: 100} (with dimensionality reduced vectors) **[Final hypothesis for Decision Tree]**	1.158	1.341	0.939	1.158

Table 8

Random forest

The random state is always set as 42.

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**
Default parameters **[Final hypothesis for Random forest]**	1.077	1.161	0.868	0.422
{min_samples_split: 0.1, max_samples: 50000, max_leaf_nodes: 16, max_depth: 80}	1.418	2.009	1.204	-1.178

{min_samples_split: 0.1, max_samples: 50000, max_leaf_nodes: 16, max_features: 2, max_depth: 80}	1.417	2.009	1.204	-1.178

Table 9

MLPClassifier

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**
random_state=1, max_iter=200 **[Final hypothesis for MLPClassifier]**	1.401	1.965	0.944	0.022
random_state=42,solver='adam', max_iter=1000	1.418	2.012	0.958	-0.001
StandardScaler(with_mean=False),random_state=42, max_iter=200	1.42	2.04	0.96	-0.01

Table 10

LSTM

The random state is always set as 42.

Model variation (parameters values)	**RMSE**	**MSE**	**MAE**	**R2**
{ epochs: 5 Batch_size: 64 } **[Final hypothesis for LSTM]**	0.788	0.620	0.580	0.633
{ epochs: 20 Batch_size: 64 }	0.821	0.674	0.603	0.607
{ epochs: 30 Batch_size: 64 }	0.836	0.699	0.620	0.567

Table 11

As the model epoch increases, the error also increases here.

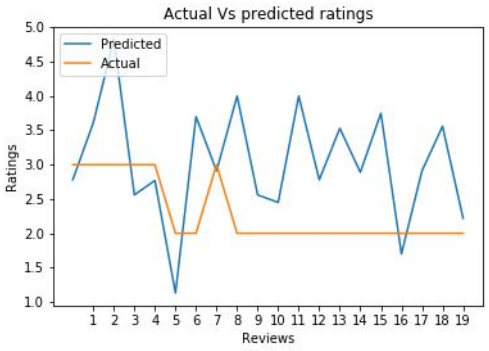
Model comparison

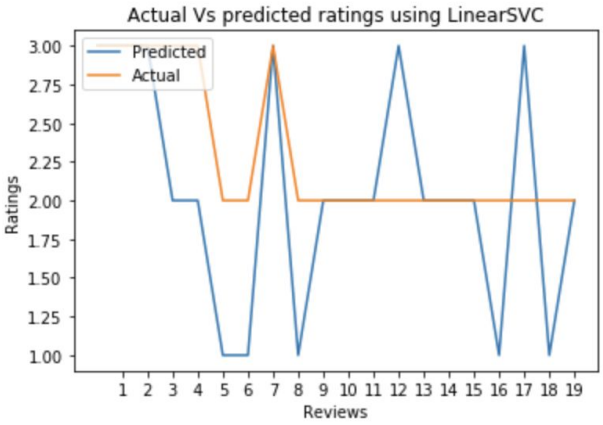
The table below shows the evaluation results on the test dataset. The model parameters that provided better evaluation errors are used for final evaluation on the test dataset.

Model evaluation on Test dataset				
Model	**Best rmse (the lower the better)**	**Best mse (the lower the better)**	**Best mae (the lower the better)**	**Best r2 (the higher the better)**
Logistic regression	0.854	0.731	0.497	0.634
Random forest	1.196	1.431	0.984	0.284
Decision tree	1.229	1.511	1.013	0.244
SGD	1.04	1.08	0.63	0.458
LinearSVC	0.912	0.831	0.536	0.584
MLPClassifier	1.41	2.01	0.99	0.01
LSTM	0.736	0.541	0.552	0.643

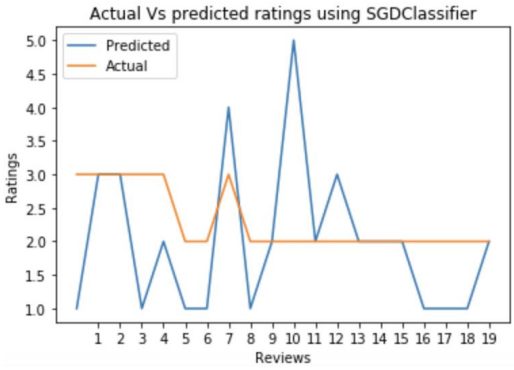
Table 12

The below table shows the plots of test data. There are around 150000 reviews on the test dataset, for clearer visualization, only a portion of 19 reviews is plotted here.

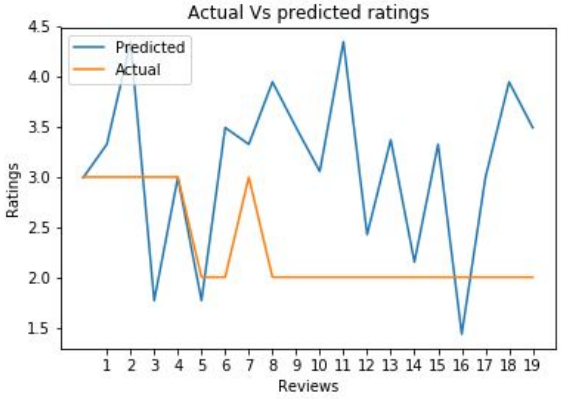
Test plots shown only for 19 reviews.	
Logistic Regression	**Support Vector machine**
	



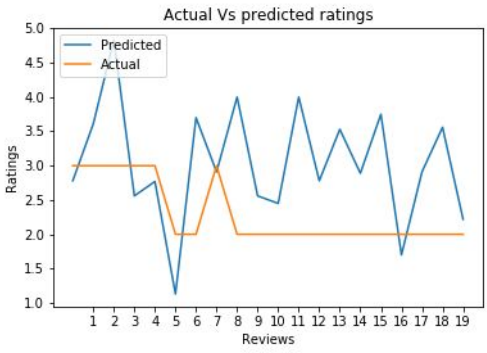
SGD classifier (SVM variant)



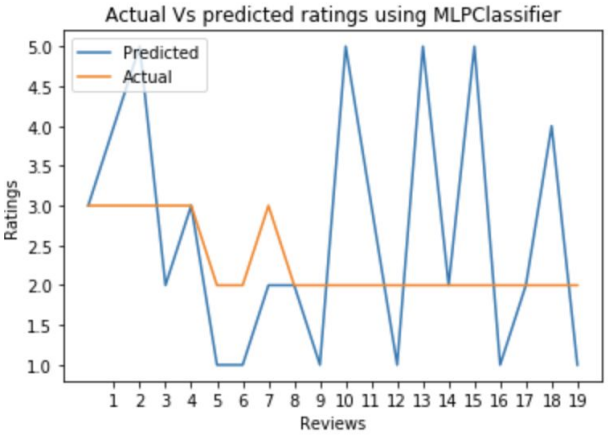
Decision Tree



Random Forest



Multi-layer perceptron



LSTM

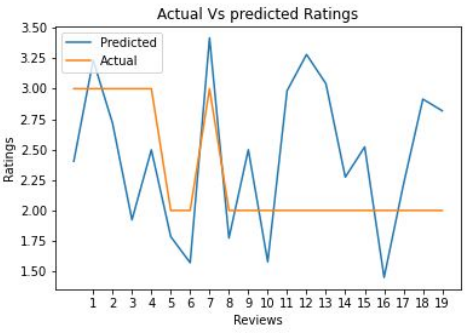


Fig 10

The below shown plot is the comparison of RMSE of all the models tried.

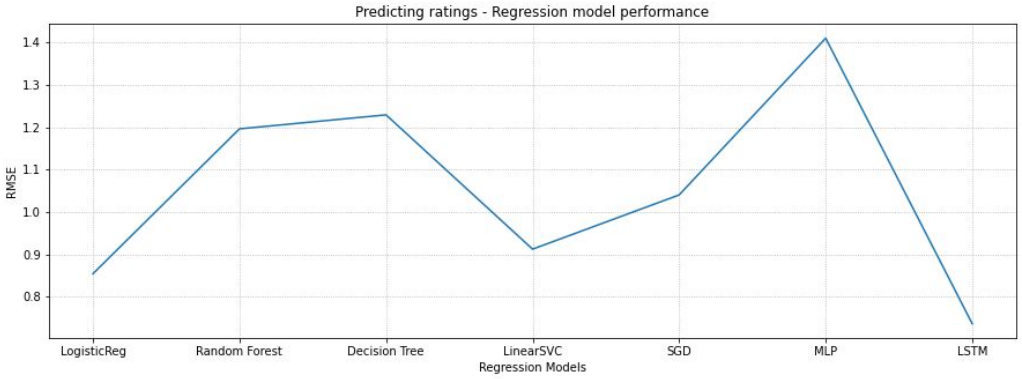


Fig 11

Result analysis

In this project, the ML models (Logistic Regression, SGD, Linear SVC, DecisionTree, Random forest), neural network (MLP) and deep learning model (LSTM) are analysed to predict ratings of the reviews. We intend to experiment different models on the big dataset and analyse its performance.

Since it is a big dataset, the performance of deep learning model on prediction is definitely better compared to other models. But in LSTM, as the number of epochs increases, the error increases as well. As a few articles suggested, it happens when the model learns on the training data which is not as representative as the test data. So, during the split, the data is shuffled and trained again as well, but there wasn't any improvement. Maybe, more experiments on creating sequential layers for LSTM is needed.

The simple models like logistic regression, LinearSVC performed better than ensemble and tree methods. Even with varying parameters, there wasn't much improvement on error in ML models. This might be because of the usage of the dimensionality reduced vectors. After applying tfidf

vectorizer, it created approx. 500k X 300k sized sparse matrix, training ML model with such a huge vector took a long time. But, the resultant vector size for each document is 300000 where it had utmost only 100 words in a sentence, which created high sparsity in the data.So, the dimensions are reduced using TruncatedSVD which is explicitly to handle sparse matrices. But the variance captured by the first principal component is just 0.004. After 15th column, it was only 0.001 for each column. This might be definitely a reason for the model's poor performance.

Even Though models has not provided satisfactory results, the learnings got from running models with different variants helped to understand do's and don'ts on the big data.

Algorithms and Tools used

Algorithms used for vectorization

TFIDFVectorizer

This vectorizer takes a set of reviews, and for each unique word, finds the inter and intra document frequency to calculate tf idf score. And, this score is used to create final vectors for the set of reviews given as an input.

GloVe Embedding

Embedding is a set of values represented as a vector. Glove embedding is a stanford's package which has embeddings for 6 Billion tokens. It provides embeddings with different dimensions such as 50d, 100d, 300d, etc. It means in a 300d file, each term has a vector of 300 values. It can be used as a dictionary for vectorizing the text data.

Algorithms used for ratings prediction

Sklearn's regression models package is used for model training. Regression models used in this project are discussed in detail below.

Logistic Regression

It is called a linear model because it predicts the output by linearly summing up the values of weighted inputs. This algorithm learns on the training data and its features, and outputs the prediction on the unknown data.

Support Vector Machines (SVM)

SVM is one of the accurate learners. It supports both dense and sparse input. For large datasets, the linearSVC works better than SVC(kernel='linear') and other svm classes.

Decision tree

This model creates a tree data structure based on the training data, which would be traversed later to make decisions on the unknown data. The leaf nodes of the tree would be used to give predictions.

Random forest

A random forest is an ensemble algorithm, which trains a set of decision trees on the samples of the data, and averages their result to make the overall strong prediction.

MLPClassifier

Multi-layer Perceptron (MLP) can be used for learning a function by training on a dataset using SGD(stochastic gradient descent) class. It consists of multiple layers such as input layer, hidden layers and output layer. It gets a set of data as an input, and passes it to multiple layers and finally predicts a single/multiple output based on the need.

LSTM

LSTM refers to Long Short Term Memory. It is a DL technique that works efficiently whenever the past information of the data needs to be reused. For text data, it definitely helps to handle ambiguous sentences.

Tools / Libraries used

MongoDB Atlas

MongoDB Atlas provides an easy way to host and manage our data in the cloud. In our project, we used the free tier cluster which provides 512 MB data storage. MongoDB Atlas also automatically handles backend administrative processes such as provisioning resources, setting up clusters, or scaling services. Connection to the database was made via Mongo compass.

command: mongodb+srv://username:password@cluster0.qkbgh.mongodb.net/

Numpy

NumPy is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.In our model it is used to perform mathematical operations on arrays , most importantly statistical routines.

Pandas

Pandas library helps to create data frames from different forms of file such as csv, json, parquet, etc., It allows storing data and manipulating it in rows of samples and columns of features.

Matplotlib

It is a visualization library that helps to plot various charts and graphs.

Seaborn

It is a visualization library that helps to plot various charts and graphs.

NLTK

NLTK is a suite of Python libraries and programs used for Natural Language Processing (NLP) for English. We have used this library for finding and removing Stop Words, Lemmatizing and tokenizing the reviews.

Sklearn

Sklearn has a set of ML packages which helps to implement the entire pipeline of building any ML model. In project, sklearn implementation of ML libraries are used for vectorization, dimensionality reduction and training the models.

Keras

Keras is an API built for creating deep learning models. It provides packages to stack up the DL model layers, to perform validation, to optimize using the objective function, metrics to evaluate the model and make predictions/classifications. In this project, the keras api are used to build a LSTM regression model.

Discussions & conclusions

Project challenges

- Huge training time (1-10 hours) and memory issues because of the large scale of data.
- MongoDB Atlas permitted only 512MB of free storage.
- Using inverted index with existing sklearn vectorizer implementations.
- Limitations of HPC usage without admin permissions.
- Data cleaning

Decisions made

- Initial yelp dataset size is preprocessed to fit in MongoDB Atlas storage limit (512 MB).
- Using SJSU HPC environment for model training which reduced the training time of some models, but still few models consumed more time (4-10 hrs).
- Reduced dimensions of the data.
- Perform initial basic data cleaning and improve on it as per the model requirements.

Final learnings

- Understanding of the NLP data pipeline.
- Handling large dataset.
- Large scale data requires higher processing power like GPU for better training time.
- understanding of dimensionality reduction techniques.
- Usage of regression ML algorithms with large scale of data.
- Hyperparameter tuning of the models.
- TfidfVectorizer's optimized performance.
- Data cleaning plays a major role in improving the model performance however well the model parameters are trained.

Future work

- Replicate the project in distributed system architecture like Spark.
- Analysis of more multidimensional indices with vectorizer performance.
- Use more deep learning algorithms to predict the ratings with hyperparameter tuning.

System requirements

All steps of the project pipeline is done in google colab with runtime as GPU other than building models. To build models, SJSU HPC environment is used. The notebooks are uploaded to github, the packages necessary to execute each notebook is described under "Necessary package installations" section with its requirements.

Pymongo[srv], pymongo[tls]	To import mongoclient from pymongo driver to connect to mongodb. Note: kernel needs to be restarted after its installation. Used only in google colab executions, haven't tried in SJSU HPC
hashedindex	To execute create_inverted_index() function which creates inverted index with terms as keys and values as the list of the documents where it occurs
keras	Installs keras API which is used to train LSTM model

Table 13

Project data files

Since the dataset is huge, all the data is stored in shared google drive. The link is given below.

https://drive.google.com/drive/folders/1uELqXJcLomwYgtXh7A0KSM0ZTaUnHCCW?usp=sharing

The following are the data files stored in the google drive and its description.

File Name	Description
yelptraindata1.pkl	cleaned training data.
yelptestdata1.pkl	Cleaned test data
yelptestlabels.pkl	Test labels
yelptrainlabels.pkl	Training labels
train_tfidf.pkl	Tfidf vectors for training data
val_tfidf.pkl	Tfidf vectors for validation data
test_tfidf.pkl	Tfidf vectors for test data
xtraindr.pkl	Dimensionality reduced training data
xvaldr.pkl	Dimensionality reduced validation data
xtestdr.pkl	Dimensionality reduced test data
lrfinal.pkl	Final better performing Logistic regression model
dtfinal.pkl	Final better performing decision tree model
rffinal.pkl	Final better performing random forest model
lstm_model	Final better performing model

Table 14

Github link

The project files are uploaded to the github.

https://github.com/anyaviswa/cmpe297-Group3

The following are the project files uploaded.

File name	Description
yelp-pipeline.ipynb	Entire data pipeline implementation except dataset preparation and model building.
yelp-Dataset-Preparation.py	A part of huge 8GB data was selected and divided uniformly into training and testing dataset to fit it in 512MB

	of MongoDB.
yelp-lstm-model.ipynb	Training, validation and evaluation of the LSTM model.
yelp-lr-rf-dt-models.ipynb	Training, validation and evaluation of Logistic regression, Random forest, Decision trees models.
yelp-svc-sgd-mlp-models.ipynb	Training, validation and evaluation of Linear Support Vector Classification model , stochastic gradient descent model, Multi-layer Perceptron classifier model

Table 15

References

1. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
2. https://pypi.org/project/hashedindex/
3. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
5. https://stackoverflow.com/questions/48202900/what-does-these-parameters-mean-in-jupyter-notebook-when-i-input-time
6. https://keras.io/api/
7. https://nlp.stanford.edu/projects/glove/