1/10/2022

Report for Masterclass in Business Analytics

Fake and Real reviews classification using Topic Modelling results

School of Business and Management
Queen Mary University of London

Yashovardhan Bhomia
STUDENT ID – 200680497
BS201334@QMUL.AC.UK

Index

Abstract

This project explores the Customer Reviews industry and its impact on decision making of the consumer. Customer reviews and ratings form an important part of the products and services industry, as it affects their sales and growth. Given how influential a review can be, it is important to have genuine and trustworthy reviews. This project explores labelling of fake or fraud reviews using Yelp's restaurant dataset. Yelp is an online platform available on web and mobile for crowd sourced reviews about various businesses. Like any other online review platform, they filter out fake reviews on their application, but their accuracy is only 16%. During the course of this project, we are evaluating the topic efficiency using topic modelling and calculating their probabilities and utilizing topic probabilities as an additional input to solve classification problem for fake and real reviews. For topic modelling we have used LDA (Latent Dirichlet Allocation) mallet model, where we get a coherence score 0.60 for 8 topics and for classification, logistic regression, K-nearest neighbor, Random Forest and Support vector machine algorithms have been used, we got highest recall score for logistic regression, 0.78

Section 1
Introduction

In the past few years, there has been a surge in choices. There’s a plethora of vendors to choose from in every industry be it product or services. To help consumers get the best out of the lot, a concept of customer reviews was introduced. These are primarily reviews from people who have experienced the product or service before. Online reviews have helped a lot in keeping the healthy competition among brands alive. They can make or break a brand image, which is why it’s so important that they’re authenticated and monitored. The biggest problem nowadays has been to address fake reviews. As these reviews are an important part of decision-making processes among customers, it has become highly important that they’re credible.

Recently all the companies have started focusing on fake reviews issues as it misleads the customers and impacts their profitability. With the help of machine learning they are trying to figure out which reviews are fake, as providing a label to each fake review manually is not possible. Detecting fake reviews is an important issue as the actual purpose of reviews is reflecting user experience and opinions, which gets impacted due to fake reviews (Crawford, 2015). Fake reviews are not always negative reviews, businesses employ the technique of writing fake reviews to boost their sales. Companies either attempt to help their sales and profits by making a misleadingly positive brand picture or by producing counterfeit negative reviews about a competitor.

These fake reviews are written either by paid human writers or are generated by machine. One of the most popular tools used for writing fake reviews is Amazon Mechanical Turk (AMT), reviews generated by it have an accuracy of nearly 90% when only the word n-gram feature is used (Luca, 2016). The AMT creators are not prone to have something similar mental perspective while composing such surveys as that of the creators of genuine phony surveys who have genuine organizations to elevate or to downgrade.

For this project we have taken data related to restaurants from Yelp, which is a platform that hosts reviews about businesses gathered from people, this includes cafes, health centers, retail shops etc. It also provides service like table reservation. Yelp is based out of the USA, with headquarters in San Francisco.

This report involves solution for fake vs real review classification, and uses output of topic modelling in doing this classification. In upcoming sections, we will have a look at the business problem which we are trying to resolve, dataset used and various pre-processing techniques applied on data. Then we will visit results from LDA model and will see the implementation of classification algorithms.

Section 2
Business Problem

Today it has become very easy for someone to go online and post reviews about a product or service. People have started using this tool as a weapon to post fake reviews online.

According to recent reports of The New York Times, there have been cases of workers being hired by businesses to post fake 5-star Yelp reviews for as little as 25 cents per review (Pontin, 2007). Some instances involved Mechanical Turk; a crowdsourcing marketplace actually owned by Amazon.

This impacts the businesses as most customers read reviews before going to a place or using a service. Online reviews are a big business for travel destinations, hotels, restaurants, service providers and FMCG (Fast moving consumer goods) products. For example, before deciding where I would like to eat tonight, I'll look at the restaurant ratings and customer reviews based on the quality of food, staff services and probably ambience. One fake review can impact my experience. So, it becomes important to be able to effectively identify a fake review be it through user account authenticity, review posting patterns or suspicious language detection.

For our problem statement we're focusing on Yelp reviews. Yelp hosts more than 70 million reviews for restaurants, mechanics, salons etc. Studies have shown that roughly 16% of reviews on Yelp are filtered (Luca, 2016). Even though the intent to filter out fake reviews is great, Yelp's filtering algorithm is not highly efficient. This results in a lot of fake reviews not getting filtered in the process. Fake reviews are not necessarily negative reviews, they could also be fake positive. The inefficient algorithm increases the risk of false positives and negatives both. For example, a genuine negative fake review might get through but a valid positive review gets filtered out as false positive.

There are a few other organizations that host customer reviews. They've also been battling with identifying and removing fake reviews from their systems. Two of the big players in this are **Trustpilot** and **Google reviews**.

Trustpilot a Danish company which hosts reviews of worldwide businesses, they allow the public to provide score and feedback according their experiences with organizations whose services they have utilized. Trustpilot charges fees to businesses for providing their services, they also invite customers to log their reviews for businesses.

They have reported that they have removed 2.2 million fake reviews in the year 2020, the vast majority of these reviews were written using automated software with human involvement (Kelion, 2021). Out of 2.2 million, 1.5 million reviews were automatically removed by fraud detection software. Trustpilot CTO informed that machine learning techniques were used for this process, where algorithm identifies multiple data like how many times a particular IP address has been used post reviews in short timeframe.

Similarly, people also provided reviews on **Google** about a particular restaurant or Hotel, Google has deployed its own automated detection system, which are built are built using machine learning algorithms.

These vetting algorithms identify content that violate the defined policies and conditions. It also combs through the data for signs of unusual activity by a user (Pritchett, 2021). For example, if there is an account that was created in India, suddenly leaves a bad review for a household services firm in Uganda and gives a poor or 1-star rating for a restaurant in Ireland. Any such identified content is instantly removed using automated models and/or flags. It is then passed on for further investigation along with validating the account's authenticity.

Google focuses on detecting content generated from click farms where fake reviews are generated. Through better identification of snap ranch movement google makes it harder to post fake substances economically, which at last makes it harder for click farms to sell reviews and bring in cash. Furthermore, to get fake business profiles before they show up on Maps, Google strengthened its Google My Business check processes with new AI models that assist with recognizing deceitful engagement. By battling huge scope endeavors to make counterfeit business profiles, Google has helped businesses so that their customers cannot be stolen by fraudsters.

Section 3
Data, EDA and Methods

3.1 Dataset

This dataset was provided by Bing Liu, author of the paper "What Yelp Fake Review Filter Might Be Doing". It was a database file, in which 2 databases were present, one for restaurant and another one was for hotel. We have considered restaurant database for this project in which there were total 3 tables present in it, review, reviewer and restaurant. So, for this project we have taken variables from all 3 tables.

The original file extracted from this database record had 788471 reviews with real reviews as 58% and fake reviews were 42%. There were NA values for few variables, count of for such variables is given below. For this project we have considered 100046 reviews due to very high computation time of actual dataset, but for this subset which we have used, distribution of real and fake reviews is remains same as that of actual dataset. Before taking the subset, we have deleted these NA values as their proportion in 788471 records is not significant.

Variable Name	Count of NA values
reviewID	1
reviewContent	2
ReviewerfriendCount	80203
ReviewerReviewCount	80203
RestaurantRating	7671

Table 1: Distribution of NA values

Below is the list of variables, their description and data type. Please note that this is not the final list of variables, we will use Topic Modelling results also as variables in classification problem.

Variable Name	Description	Data Type
date	Date on which review was posted on Yelp	object
reviewID	ID generated for Review	object
reviewerID	ID of person posting the review	object
reviewContent	Review posted by reviewer about restaurant	object
rating	Rating given by reviewer to restaurant	int64
usefulCount	Count of customers which have found review as useful	int64
coolCount	Count of customers which have found review as cool	int64
funnyCount	Count of customers which have found review as funny	int64
flagged	Posted Review is Real or Fake, 'N' is for Real and 'Y' is for Fake	object
restaurantID	ID of restaurant receiving the review	object
ReviewerfriendCount	Number of friends of Reviewer	float64
ReviewerReviewCount	Number of Total reviews posted by a Reviewer	float64
RestaurantRating	Overall Rating of Restaurant	float64

Table 2: Variables and their data types

Target variable for Topic modelling is “reviewContent”.

Target variable for classification is “flag”, and which was created using one-hot encoding to classify ‘N’ as 0 and ‘Y’ as 1 respectively, original variable name was “flagged”.

Feature variables for classification are “reviewContent”, “rating”, “usefulCount”, “coolCount”,” funnyCount”,” ReviewerfriendCount”, “ReviewerReviewCount”,” RestaurantRating”

Please not that these are not the final feature variables for classification, Topic probabilities for 8 topics will also be used from output of topic modelling.

3.2 EDA

Below is the summary statistics of the Dataset: -

```
------------------------------------------------------------------------------------------------------------------
Status from Basic Observations
------------------------------------------------------------------------------------------------------------------

There are 100046 Observations and 14 Features in the Dataframe.

There are 11665 Unique Users in the Dataframe.

There are 59770 Unique Restaurants in the Dataframe.

There are 58360739 Lemmatized Terms from the Reviews Column.

The Longest Review Comprises 4030 Lemmatized Words.

The Shortest Review Comprises 0 Lemmatized Words.

Number of Real Reviews (Flagged as N) and Fake Reviews (Flagged as Y):
N    56853
Y    43193

------------------------------------------------------------------------------------------------------------------
```

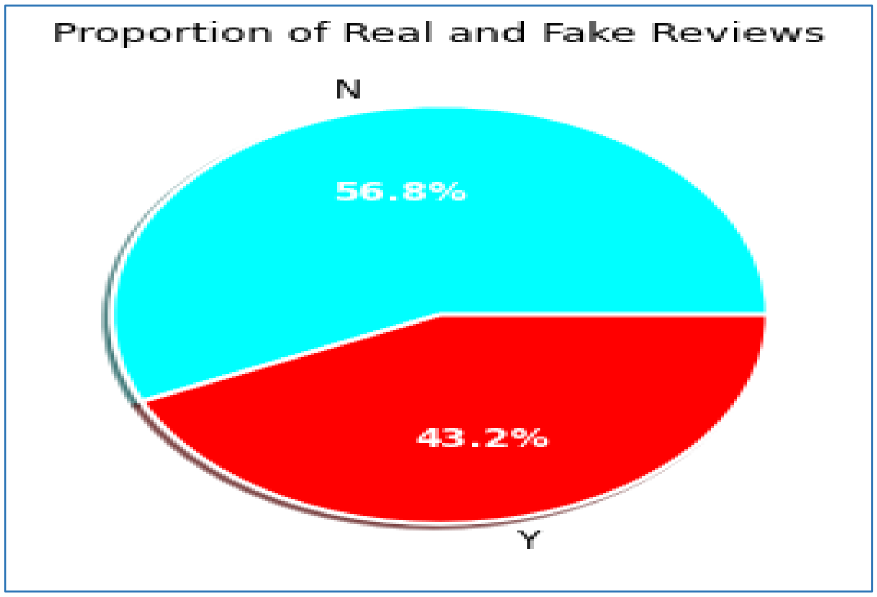


Figure 1 - Proportion of Real and Fake Reviews

The above graph displays the distribution of real and fake reviews using flagged variable, “N” means that reviews are not flagged and are Real, there are 57% real reviews in dataset and “Y” means that reviews are flagged and are fake, there are 43% fake reviews.

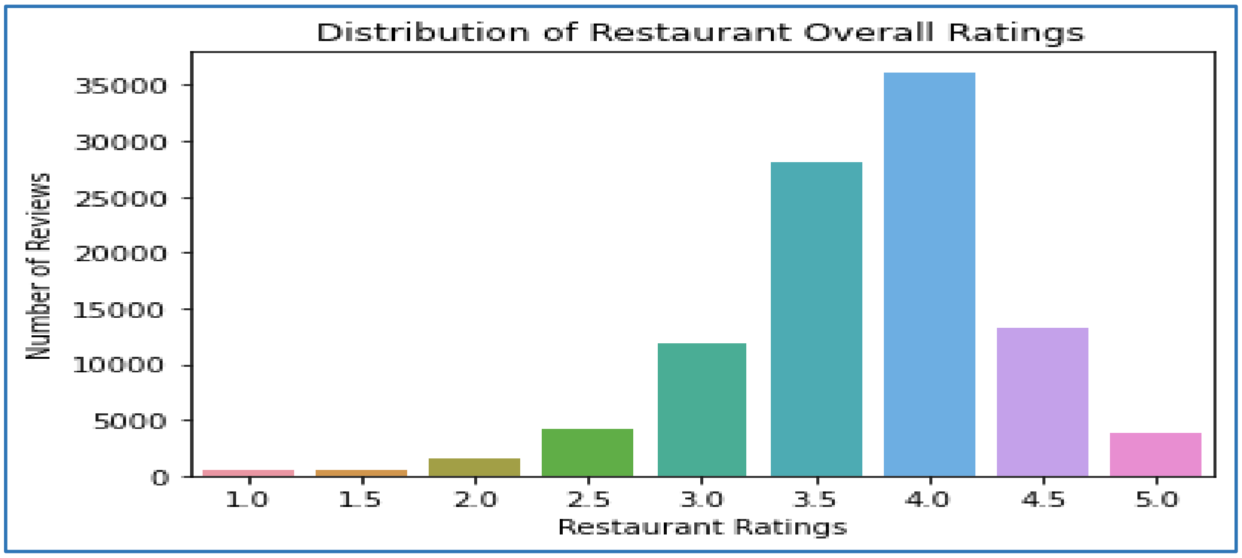


Figure 2 – Restaurant overall ratings

In above graph we can see the distribution of restaurant Overall Ratings, there are around 35k such reviews for restaurants which have overall rating as 4. Number of reviews posted for restaurants having ratings below 3 stars are less than 5000. A key observation is that 5-star rated Restaurants have very a smaller number of reviews, they also have less than 5000 reviews.

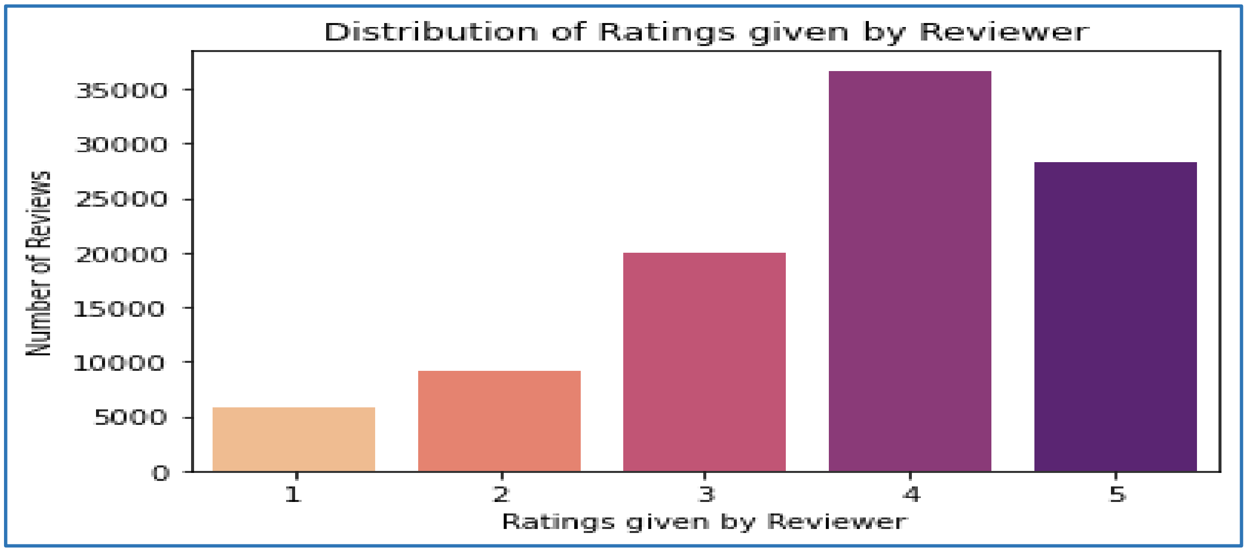


Figure 3 – User Ratings vs Number of Reviews

In figure 3, maximum number of reviews giving rating as 4 is approximately 35000 and around 27000 reviews have given 5-star ratings to restaurant. But we have very less reviews giving rating less than 3, collectively such reviews are around 35000.

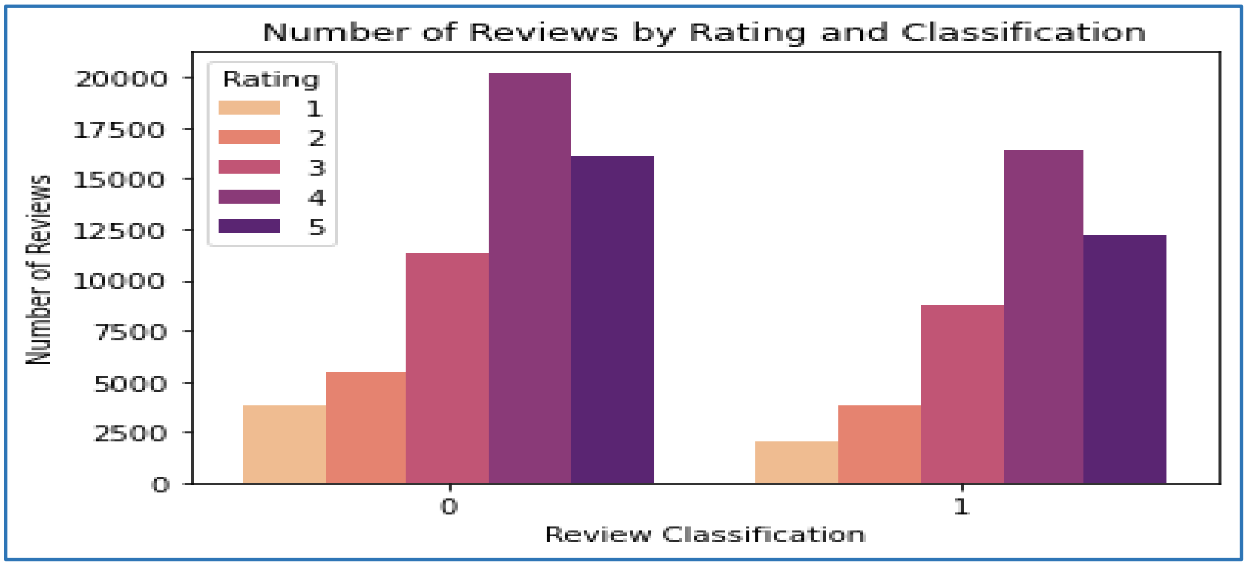


Figure 4 – Number of reviews vs number of ratings (classified as fake and real)

In Figure 4, on x-axis 0 represents Real review and 1 is for fake reviews. We can compare and see the distribution of ratings, which are separated for fake and real reviews. For 5-star rating, real reviews are around 16000 (approx.) and for fake, it's around 12000 (approx.). Rating 1 and 2 have kind of similar distribution, but major difference lies for 3-star and 4-star ratings, we have more reviews labelled as real for these two ratings as compared to fake reviews.

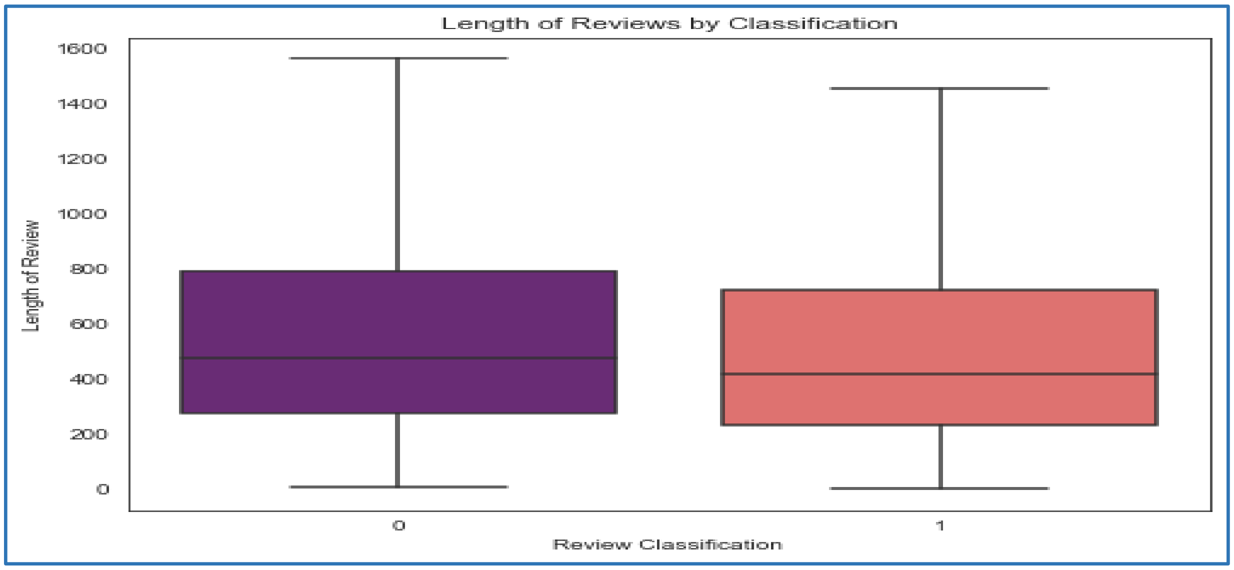


Figure 5 – Length of review for Real (0) and Fake (1) reviews

In figure 5, we compared the length of reviews for fake (1) and real (0), there is not a huge differentiation among these reviews based on length of review, both reviews are using similar length, so we can say that by looking at length it will be hard to determine the fake and real reviews.

Now let’s look at few word clouds, they generally provide an idea about most frequent words used in text, so we compare the word cloud for reviews based on ratings and real and fake classification.

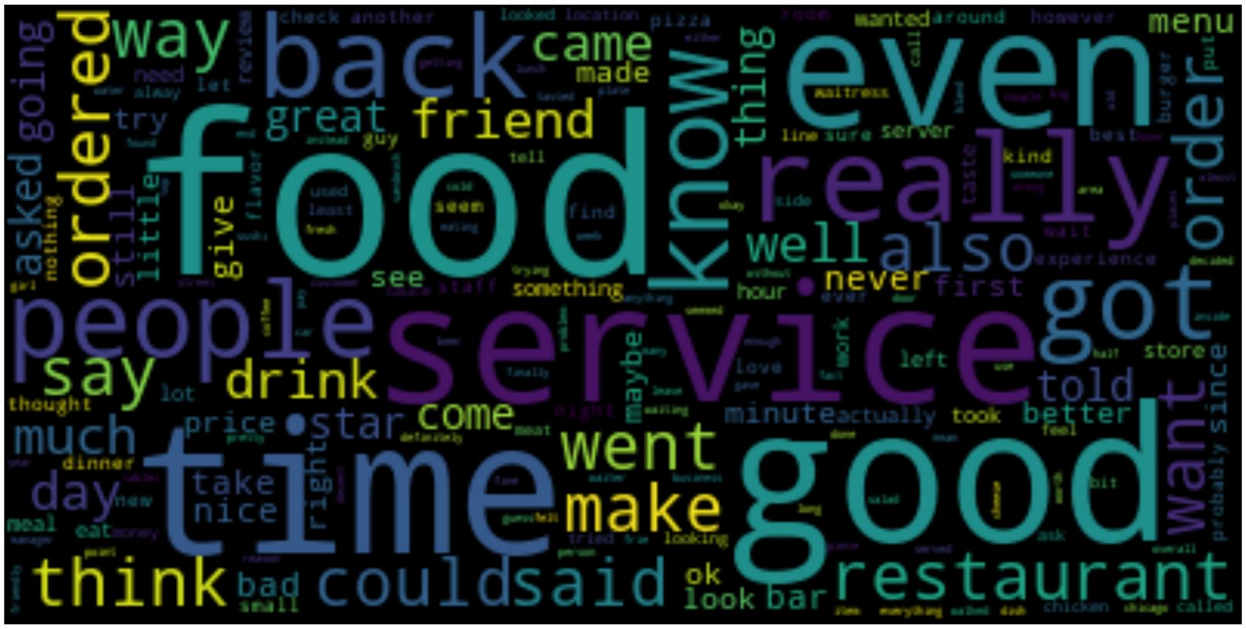


Figure 6 - Reviews with Positive Rating (with ratings >3)

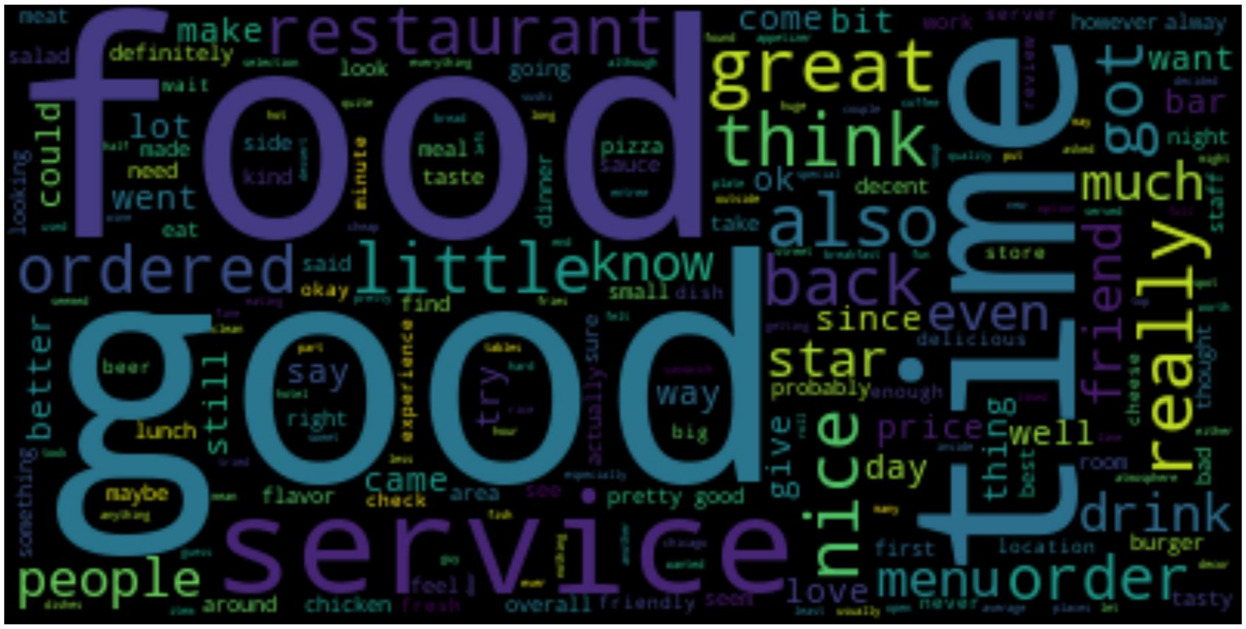


Figure 7 - Neutral Reviews (with Ratings = 3)

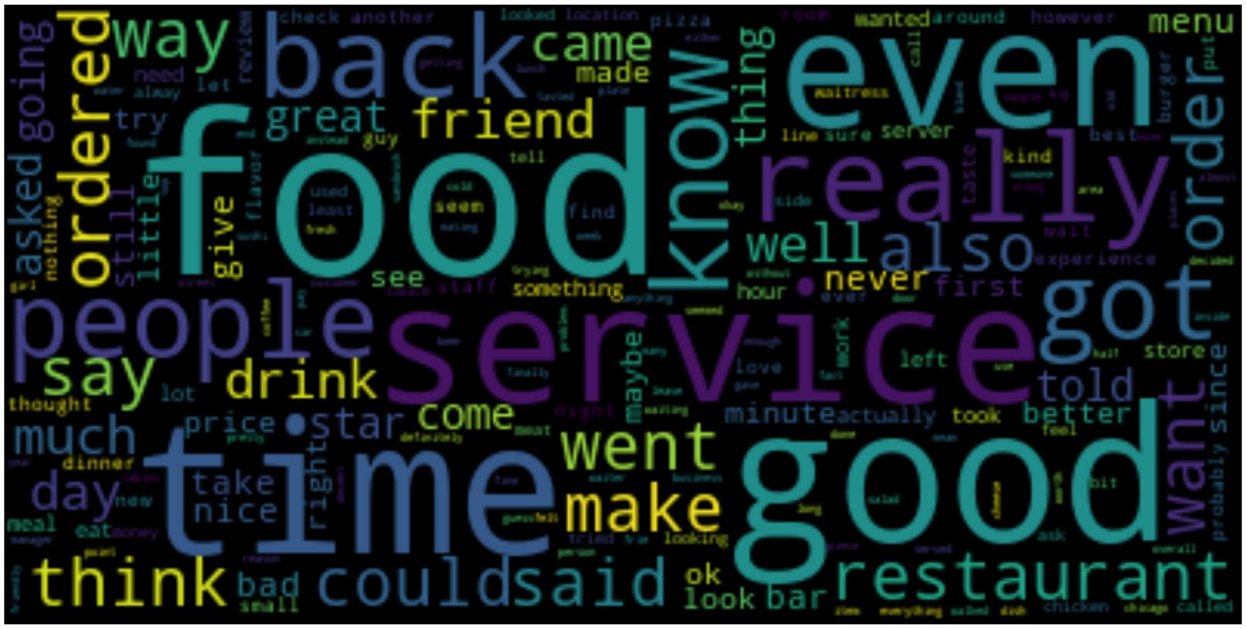


Figure 8 - Reviews with Negative Ratings (User Rating < 3)

Most frequent words used by customers are similar, irrespective of what rating they have given with their review.

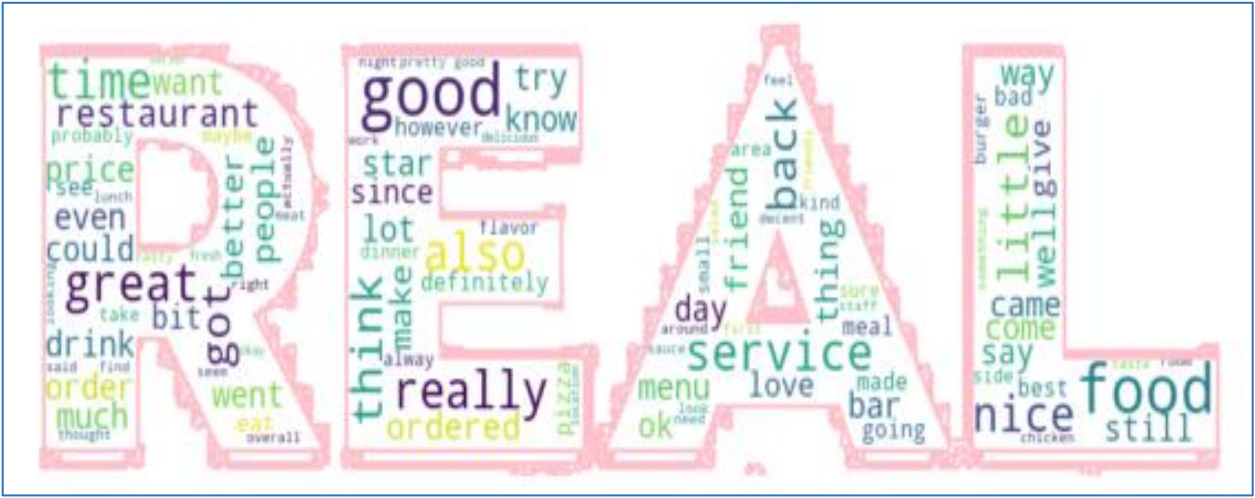


Figure 9 - Word Cloud for Real Reviews

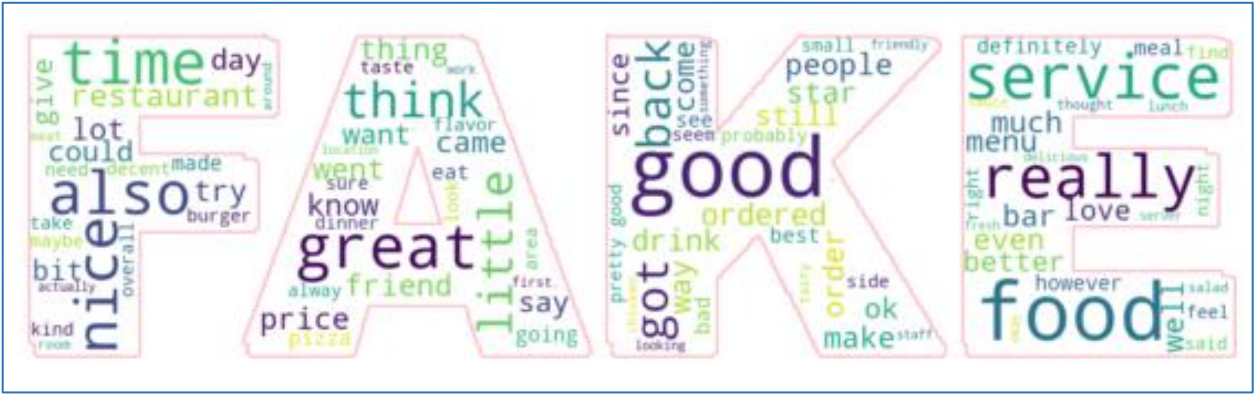


Figure 10 - Word Cloud for Fake Reviews

In All the both the word cloud for Real and Fake Reviews, we can see that similar word patterns have been used, so it's not possible for a person to tell the difference between genuine or fake review. This leads to bigger problem of businesses getting impacted due these fake reviews, as customer can pass a judgement that whether a restaurant is good or not and will impact the sales of that restaurant.

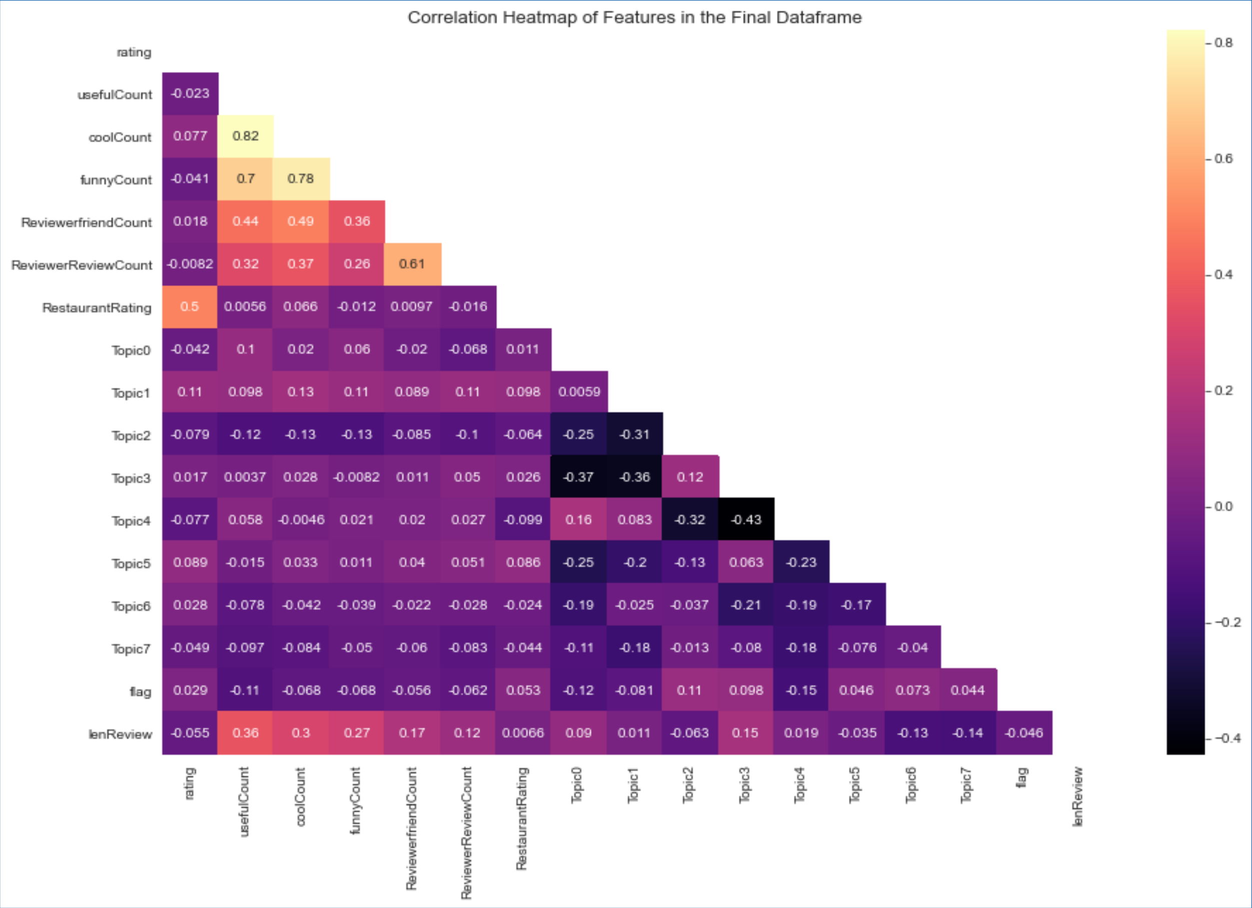


Figure 11 – Correlation matrix

The correlation matrix is including the Topic variables in which probability values are stored, they are labelled as Topic0 to Topic7.

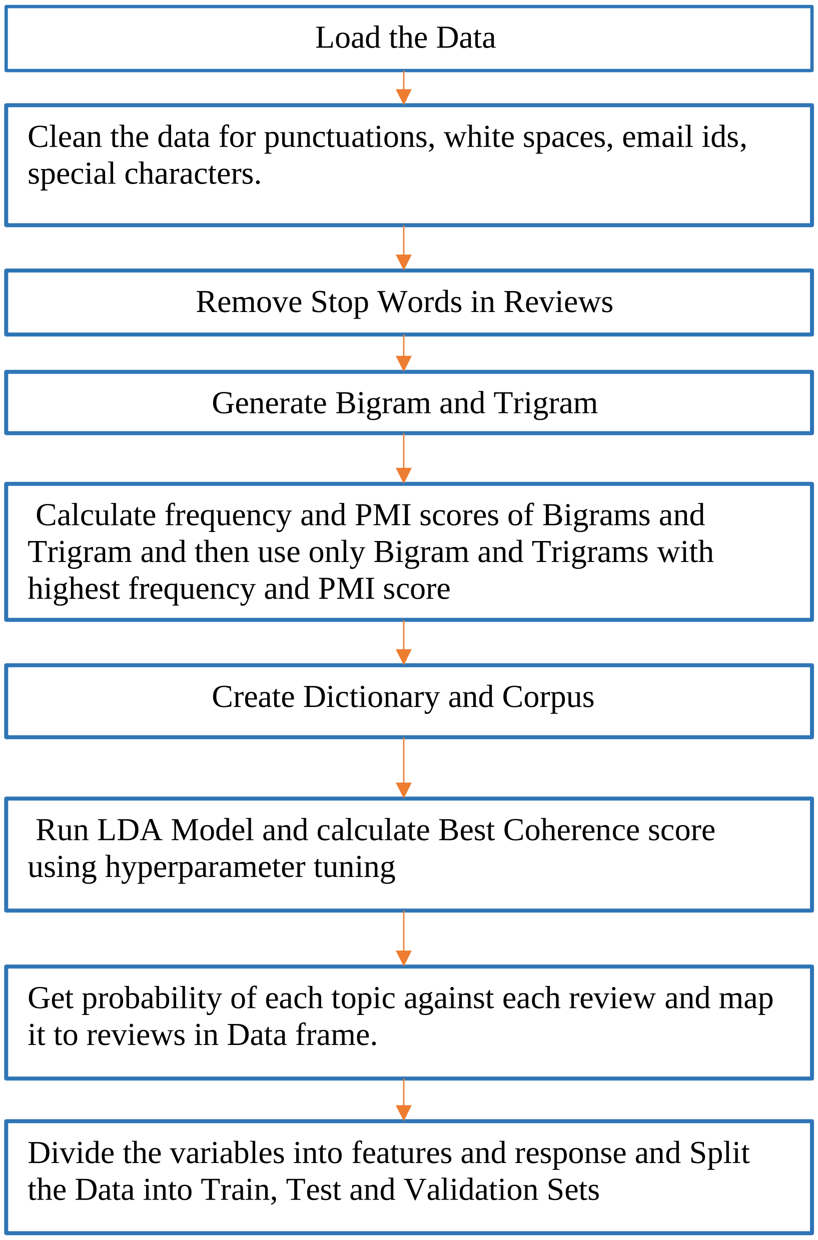
We can see that there is very high correlation among usefulCount, coolCount and funnyCount, so in our classification problem, we have dropped coolCount and funnyCount and considered only usefulCount as including variables having correlation among them create unnecessary noise in results and they will be biased. If highly correlated variables are not removed, it becomes difficult for machine learning algorithm to gauge relationship between each response variable and the feature variables independently, as high correlated feature variables create a situation where if there is a change in one variable, it automatically triggers changes in the other variable.

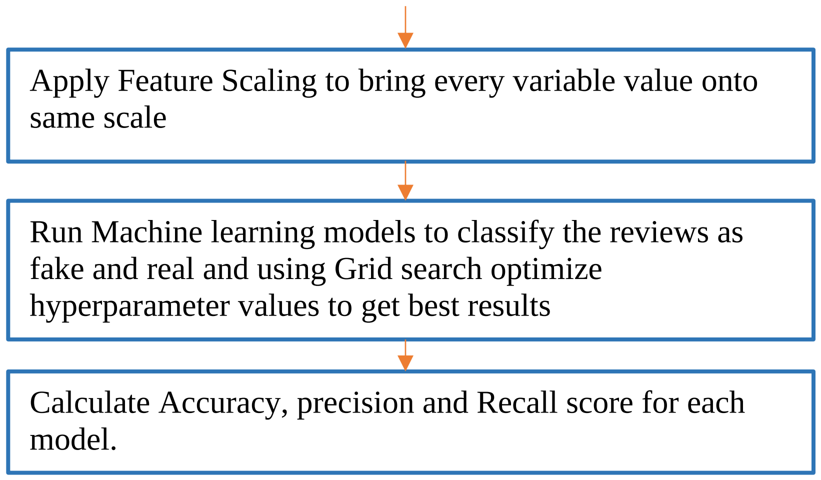
3.3 Methods

For classifying reviews as fake or real, we will use Topic Modelling to get the most frequent topics and then we will find out probability of each topic against each review and then use that as input to classification algorithms.

Latent Dirichlet Allocation (LDA) algorithm has been used for topic modelling which utilizes Python's genism package. The problem is to find good quality topics that provide meaningful and clear information. Extracting topics is dependent on how we pre-process and clean the text before feeding it into LDA Model. Now let's look at the pipeline/steps which has been used to extract the useful topics and how topic modelling was linked to classification problem.

FLOW CHART OF PROJECT


Load the Data
Clean the data for punctuations, white spaces, email ids, special characters.
Remove Stop Words in Reviews
Generate Bigram and Trigram
Calculate frequency and PMI scores of Bigrams and Trigram and then use only Bigram and Trigrams with highest frequency and PMI score
Create Dictionary and Corpus
Run LDA Model and calculate Best Coherence score using hyperparameter tuning
Get probability of each topic against each review and map it to reviews in Data frame.
Divide the variables into features and response and Split the Data into Train, Test and Validation Sets


Apply Feature Scaling to bring every variable value onto same scale
Run Machine learning models to classify the reviews as fake and real and using Grid search optimize hyperparameter values to get best results
Calculate Accuracy, precision and Recall score for each model.

Data-Preprocessing Steps: -

Step 1: We have made use of **Stop words** library which is imported from NLTK. It's basically a collection of pre-defined words which do not pass meaningful information about the text. Few examples of Stop words are I, me, We, can, would. This Step will remove these words from reviews.

Step 2: Removing punctuations, white spaces, email ids, brackets, special characters from reviews. These only create distraction and obstacle in getting good results.

Step 3: Next Step was to use only **Noun** type structures out of the cleaned data so verbs and unnecessary words can be avoided, as they will not pass any important information about Restaurant and their services.

Step 4: Next Step is to generate **Bigram** and **Trigram**; Bigram and Trigram are sequence of two and three words repeated frequently. Doing this step will help in extracting the key words which are used frequently by customers while writing reviews.

Step 5: Then we will calculate **frequency** and **PMI** (Pointwise Mutual Information) score for Bigram and Trigram. The goal of PMI is to measure the possibility of two words appearing together, while taking into account that it could be caused by the frequency of the single terms. We have only considered Bigrams and Trigrams with PMI score greater than 5 and chosen top 500 out of them by setting threshold for their frequency.

Step 6: Now we will create **Dictionary** For each review reporting how many words and how many times those words appear and then create a Corpus, which is a mapping of (word_id, word_frequency). Then these **Corpus** and Dictionary will be used as an input to LDA model.

Step 7: We run LDA model now and based on **Coherence** and **Perplexity** score and by avoiding overlap of topics we consider the count of Number of Topics.

Step 8: Then we have calculated the **probability** of each topic for each review and mapped it in Dataframe. Dataframe of probability results and dataframe having original dataset are merged.

Step 9: We calculate length of each review and using the merged dataframe we divide variables into features and response variable. After that we split the features and response variable into train (60%), test (20%) and validation (20%).

Step 10: Not all the variables are on same scale, for example, variables scoring probabilities are in range 0 to 1, but length of review, rating given by user, restaurant rating, usefulCount, ReviewerfriendCount, ReviewerReviewCount, RestaurantRating are on different scale with values going into thousands. So, for brining every variable on same scale, **minimum maximum scalar** has been used with now all variables having range of values between 0 and 1.

Step 11: We run classification algorithms like logistic regression, SVM etc. and compute the results for accuracy, precision and recall.

Models used:

For Topic Modelling, we have used LDA and LDA mallet model, Gensim has a wrapper that allows us to use Mallet's LDA from within. For using it we need to download the file and point model mallet to the directory. Mallet has a well-functioning LDA implementation. It runs faster and separate topics more effectively, than LDA model. Mallet is basically as java based package written by Andrew McCullum which is used for topic modelling.

For classification we have used logistic regression, random forest, SVM and K nearest neighbor.

Logistic Regression is a model for binary and linear classification, it is very easy to realize and achieves very good performance with linear separable classes. It is a process of modeling the probability of a discrete outcome given an input variable. Since our reviews are flagged as 0 and 1, it’s important to use logistic regression.

K nearest neighbor is a type of supervised learning technique which is used for classification and regression problem, it assumes that similar things (sharing same properties) exist in close proximity. It considers K Nearest Neighbors (Data points) to predict the class or continuous value for a new Datapoint. It’s helpful for binary classification as it will assume that 0 and 1 data points are situated near to each other.

Random Forest is an algorithm based on combination of decision trees and can be used for both classification and regression tasks, its usually trained using bagging method, where the idea is that a combination of learning models will increase the overall results. While growing the trees, random forest adds randomness to the model. Main point is that it does not look for most important feature while splitting a node for decision tree, instead it looks for best feature among a random set of features.

Support vector machine (SVM), is also used for both classification and regression tasks, it produces high accuracy with lesser computational time. The aim of SVM is to find a hyperplane in a N-dimensional space, where N is the number of features, that distinctly classifies the data points.

Below is the list of Machine learning libraries used in this project:

ML Libraries used	Description
NumPy	A library for handling arrays and matrices with pre-defined set of mathematical functions.
pandas	Used for data cleaning, analysis and manipulation of multi-dimensional array.
gensim	A library for processing large set of raw and unstructured text.
pyLDAvis	Helps in visualization of topics extracted from series of text.
matplotlib	It’s used for creating static and animated visualizations.
seaborn	A library for creating statistical graphs for univariate and bivariate data and is built on top of matplotlib
sklearn	It’s a library which contains tools for machine learning models including classification and regression.

Table 3: Machine learning libraries

Section 4
Analysis and Results

We have divided the analysis and results part into two sections, we will first look at the results of topic modelling and then we will cover classification problem.

4.1 Toping Modelling using LDA

After dictionary and corpus was created by steps mentioned in previous section, it was used as an input to LDA model, we have run both LDA and LDA Mallet model and found that Coherence score was better for mallet, we have considered number of Topics as 8, as after that coherence score is only slightly high but we observe high overlapping topics. LDA has score of 0.54 for 8 topics and mallet has 0.60 score for 8 topics, so clearly mallet is able to do better topics identification.

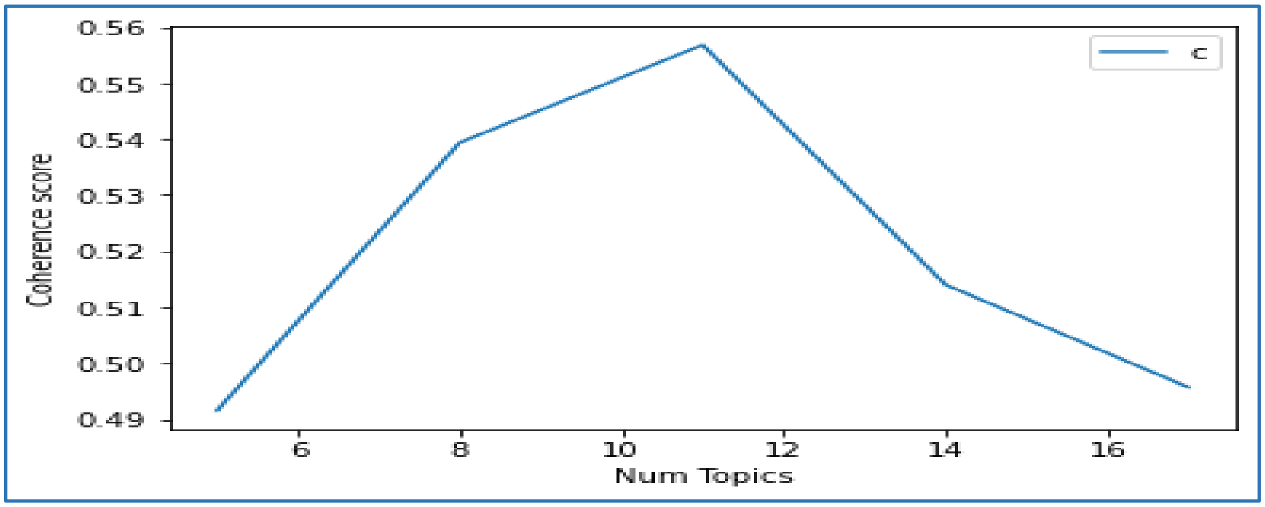


Figure 12: Coherence score for different number of Topics for LDA Model

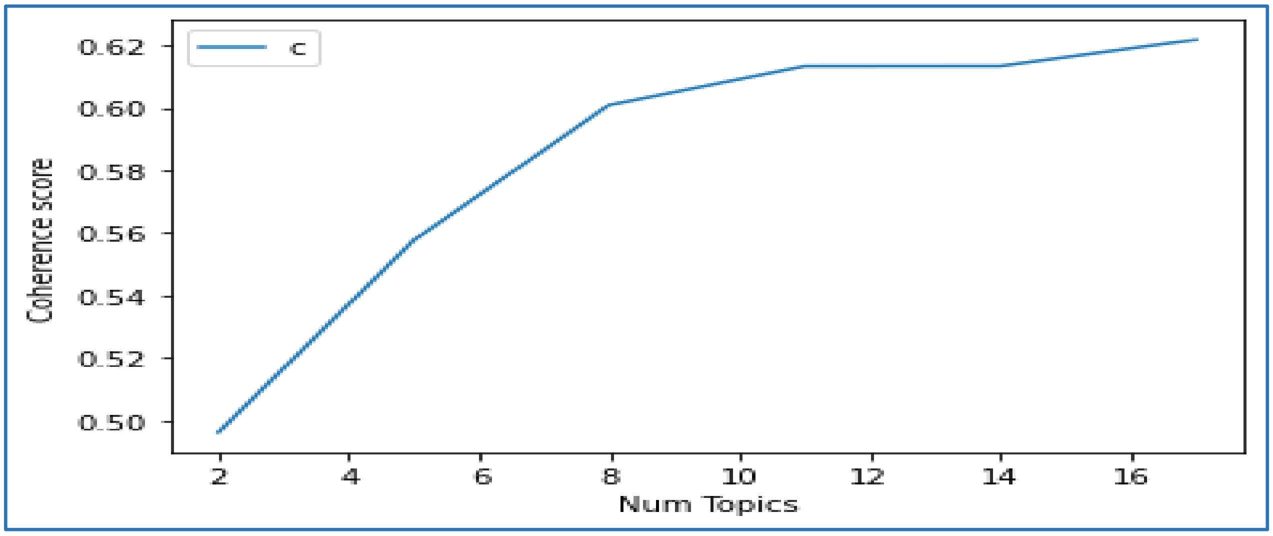


Figure 13: Coherence score for different number of Topics for LDA Mallet Model

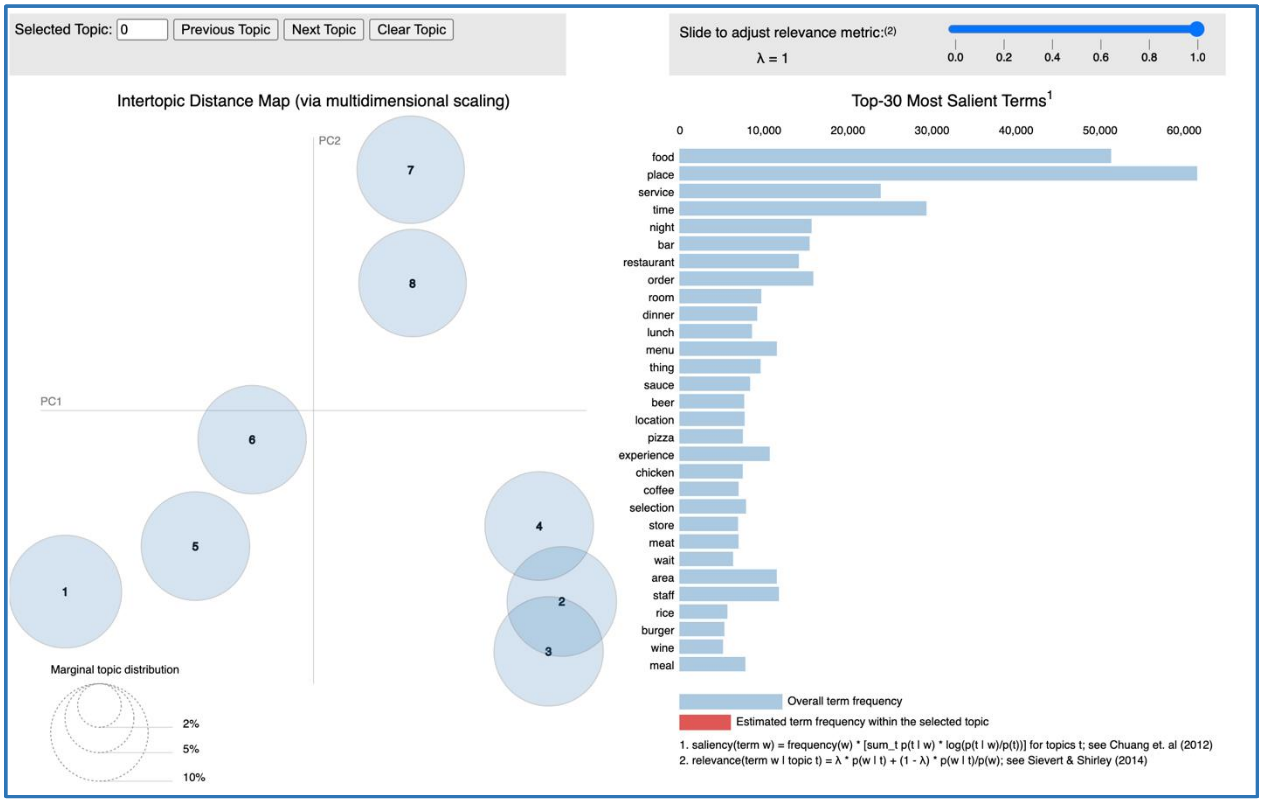
In topic modelling, the **coherence score** is used to determine how human-interpretable the subjects are. A topic is created by N-words with high probability of belonging to that particular topic. The coherence score is a measurement of how similar these words are to one another. Higher coherence score should be considered but at the same time we need to see that there should not be very high overlapping of topics with each other.

Also, if we compare the perplexity score for both the models, LDA model has a score of -6.2437 and LDA mallet has -7.5192, so we will prefer the mallet model as its perplexity score is lower. **Perplexity** is a statistical measure equivalent of geometric mean, so a lower perplexity score means that topic identification is better.

Measure	Value
Coherence score	0.6011
Perplexity score	-7.5192

Table 4: Scores for LDA mallet model

Below is the visualization of Topics we found from LDA mallet model, there is only slight overlapping between them, so results look fine.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 12 – Topic Visualization of LDA output

Now let’s look at the word distribution in each topic, with their weightage value in that topic:

```
[(0,
  '0.069*"time" + 0.030*"experience" + 0.021*"work" + 0.019*"day" + 0.018*"staff" + 0.017*"review" + 0.016*"week" + 0.015*"job" + 0.013*"owner" + 0.011*"care"'),
 (1,
  '0.019*"day" + 0.019*"city" + 0.017*"lot" + 0.016*"year" + 0.014*"water" + 0.013*"part" + 0.013*"town" + 0.012*"trip" + 0.011*"time" + 0.011*"weekend"'),
 (2,
  '0.211*"food" + 0.091*"service" + 0.058*"restaurant" + 0.038*"dinner" + 0.036*"menu" + 0.035*"lunch" + 0.022*"quality" + 0.022*"order" + 0.019*"sushi" + 0.018*"meal"'),
 (3,
  '0.032*"sauce" + 0.029*"chicken" + 0.027*"meat" + 0.022*"rice" + 0.019*"flavor" + 0.019*"salad" + 0.017*"bread" + 0.016*"beef" + 0.016*"spicy" + 0.016*"soup"'),
 (4,
  '0.040*"room" + 0.032*"location" + 0.028*"store" + 0.020*"staff" + 0.017*"price" + 0.016*"car" + 0.016*"hotel" + 0.014*"door" + 0.014*"check" + 0.014*"line"'),
 (5,
  '0.029*"coffee" + 0.021*"bit" + 0.020*"breakfast" + 0.018*"chocolate" + 0.015*"sandwich" + 0.015*"side" + 0.014*"taste" + 0.011*"tea" + 0.010*"love" + 0.010*"variety"'),
 (6,
  '0.111*"place" + 0.066*"night" + 0.065*"bar" + 0.032*"beer" + 0.031*"selection" + 0.027*"area" + 0.022*"burger" + 0.022*"wine" + 0.020*"music" + 0.019*"group"'),
 (7,
  '0.148*"place" + 0.044*"order" + 0.041*"time" + 0.037*"thing" + 0.032*"pizza" + 0.027*"wait" + 0.024*"kind" + 0.018*"eat" + 0.017*"half" + 0.016*"lot"')]
```

For example, if we look at Topic 0, we can see that highest weightage is of time (0.069) and then other words are present in decreasing order of their weightage.

Next step was to find probability of each topic against review, so that we can use those values as an additional input to our classification algorithms, for that we have used get_document_topics function on mallet model results.

4.2 Fake vs Real review classification by adding Topic probabilities

For classification, we have used total 4 algorithms, whose results are displayed in table below and we did hyperparameters tuning using grid search.

Grid search, basically creates a grid to test all the combination of hyperparameters to get the best values for each parameter of the model, using grid search we tried to tune and get the best parameters value for SVM and random forest.

For random forest, we got values for n_estimators =200, min_samples_split = 2, max_depth=25, where n_estimators represent number of decision trees in the forest, min_samples_split is for the number of splits at each node of the tree, max_depth tells how big a tree can grow, increasing the depth means increasing the possible feature combination, which will help in capturing more information about the data.

For SVM, the best hyperparameters value we got were C=10 and gamma = 1, where C parameter adds a penalty for misclassified data point, if C is small, penalty is also low and gamma parameter basically controls the distance of influence of a single training point, low value means that it will create a large radius based on similarity and more data points can be clubbed together.

But the best result we have got below is from logistic regression which does not has any hyper parameters, so tuning was not required for it.

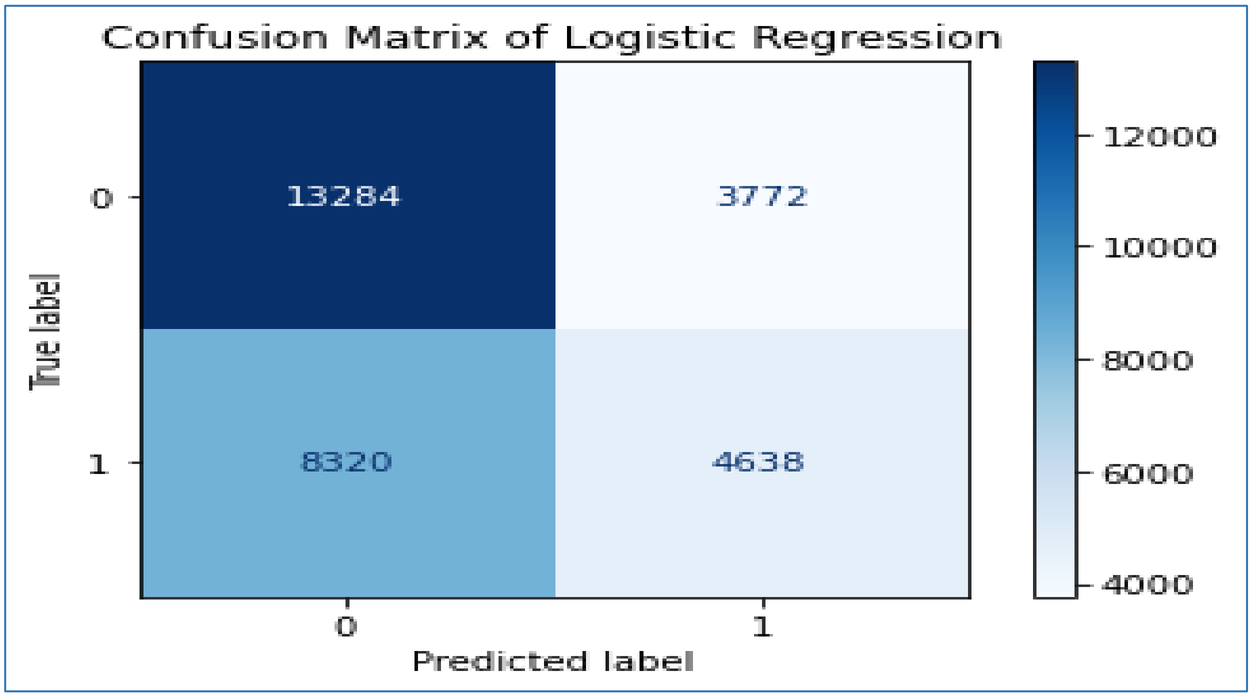


Figure 13: Confusion Matrix for Logistic Regression

Models Results:

Model	Accuracy Score	Precision	Recall
Logistic Regression	59.71%	0.61	0.78
K nearest neighbor	58.80%	0.63	0.65
Random Forest	63.12%	0.67	0.7
SVM	63.01%	0.66	0.72

As our main is to identify and label fake reviews we will consider Recall Score and we will be ok if any Real review is mapped as Fake review, but any fake review should not be labeled as fake. The highest Recall Score we are getting for Logistic Regression, which is 0.78

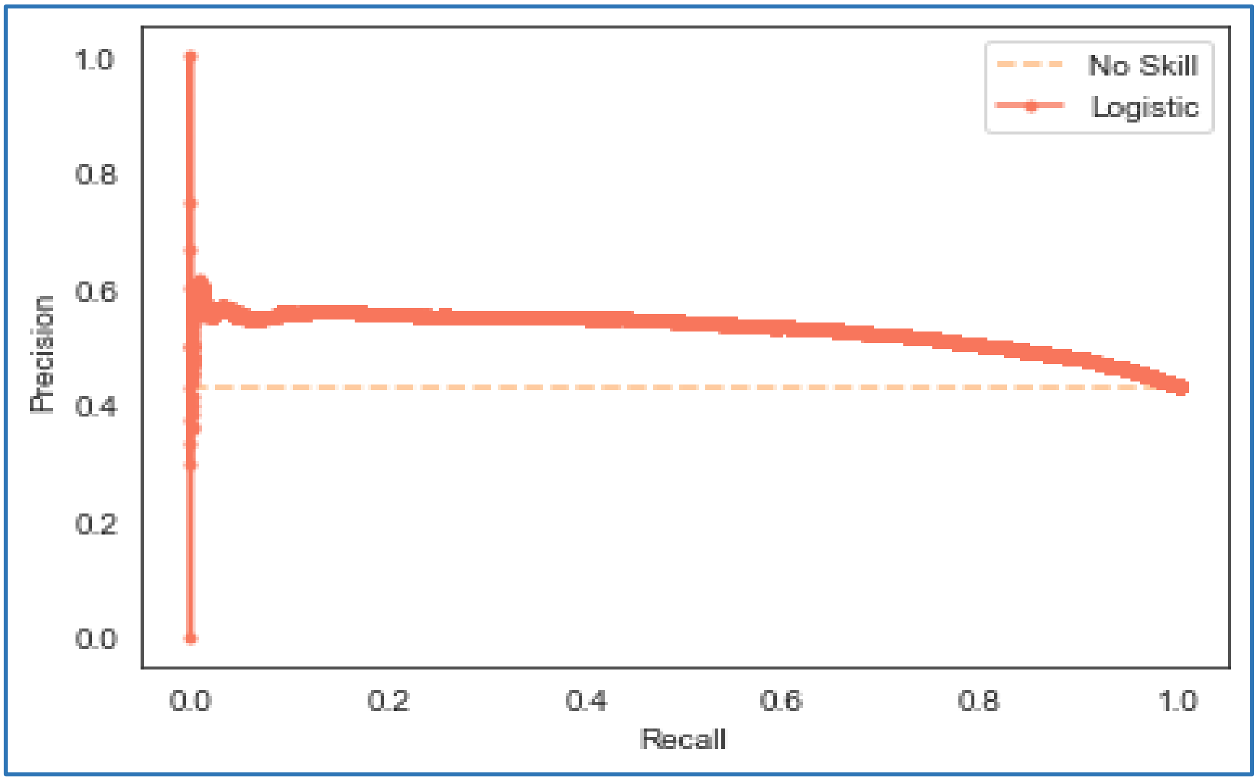


Figure 14: Precision –Recall curve

The precision-recall curve for logistic regression is displayed above, with thick orange line and a base line classifier is shown with orange dots. It indicates that logistic is giving high scores for precision and recall as compared to base classifier. It fits the logistic regression model on training dataset and uses that to make prediction on test set.

Section 5
Discussion and Conclusion

The results gathered from the project have taught us key aspects about fake and real reviews posted by customers. However, it will be interesting to see the future aspects of this business problem. As the internet continues to grow, the quantity and impact of online reviews is increasing day by day, so it becomes important for a firm to categorize which review is fake and which is real. Our solution will help Yelp in labelling suspicious reviews as fake, as fake reviews impact the business of restaurants. Not only do they create a negative or positive image in customer's mind, but also starts impacting their sales and profitability. So, Yelp will only keep real reviews on their website and fake ones will be deleted.

This will help in increasing their credibility in market and will boost their brand value as more customers will visit Yelp as they will find only authentic reviews about a place of interest.

Limitation of this report is that there was very high computational time for processing the original dataset which comprises of 788471 records, it was taking a whole day just to compute initial modelling, so it was not possible for us to go ahead with complete dataset, as for testing small change in pre-processing and hyperparameter tuning we needed to run the complete code.

In dataset we should have variable for recording user account activity, generally fake reviews are created by new accounts, so we can store that information in data set to see which accounts perform suspicious activities.

Yelp should also capture source Internet Protocol (IP address) for each activity, this will help use to determine which IP is being used when a series of reviews are posted in very short span of time. Generally, people posting fake reviews use different accounts but if they are generated from same IP address, then we can narrow down such reviews.

Also, the dataset should have equal distribution of real and fake reviews among each user rating, for example total reviews (fake + real) giving 5-star ratings is 10000, so same distribution should be followed for other user ratings also. These will help in understanding the sentiments behind giving such ratings and will also improve the fake vs real classification modelling results.

Also, we can go further and study the implementation of Neural Network, CNN and LSTM on the classification problem, it's based on deep learning algorithm which tries to find high level abstractions in data.

Also, we can look at other methods for getting topics out of raw text. There are neural network models like BERT which are effective against handling of natural language-based data.

So, to conclude, we can say that we have got some decent results for labelling fake reviews using topic modelling but there is a scope of further studies using deep learning methods, which can be applied both to process raw data and classification problem.

Bibliography

Crawford, M., 2015. Survey of review spam detection using machine learning technique. *Journal of Big Data.*

Kelion, L., 2021. *Trustpilot removed 2.2 million bogus reviews in 2020.* [Online]
Available at: https://www.bbc.co.uk/news/technology-56100082

Luca, M. a. Z. G., 2016. Fake it till you make it: Reputation, competition, and Yelp review fraud. *pubsonline.informs.org.*

Pontin, J., 2007. *New York Times.* [Online]
Available at: https://www.nytimes.com/2007/03/25/business/yourmoney/25Stream.html

Pritchett, D., 2021. *A look at how we tackle fake and fraudulent contributed content.* [Online]
Available at: https://blog.google/products/maps/google-maps-101-how-we-tackle-fake-and-fraudulent-contributed-content/

Appendix

BUS131_Peer-Assessment-

```
######## Importing libraries #######
pip install pyLDAvis
pip install gensim
pip install -U spacy
import nltk; nltk.download('stopwords')
import re
import numpy as np
import pandas as pd
from pprint import pprint
import re
import string
##### Gensim ####
import gensim
#pip list
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

##### spacy for lemmatization #####
import spacy

###### LDA Plotting tools ######
import pyLDAvis
import pyLDAvis.gensim_models
import matplotlib.pyplot as plt
```

```
%matplotlib inline

from nltk.corpus import stopwords

######## word cloud library #########
from wordcloud import WordCloud
from wordcloud import ImageColorGenerator
from PIL import Image
import seaborn as sns

import seaborn as sns



########### libraries required for classification ###########
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from collections import Counter
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve, precision_recall_curve, auc, make_scorer, precision_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import plot_confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve, precision_recall_curve, auc, make_scorer, precision_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from matplotlib import pyplot



stop_words = stopwords.words('english')



stop_words.extend(["i", "you", "we", "us", "they", "them", "she", "her", "he", "his", "him", "himself",
"herself","she's", "he'd", "she'd", "he'll", "she'll", "shan't", "mustn't", "when's", "why's", "how's","a's", "ain't", "aren't",
"c'mon", "c's", "can't", "couldn't", "didn't", "doesn't", "don't", "hadn't", "hasn't", "haven't", "he's", "here's", "i'd", "i'll",
"i'm", "i've", "isn't", "it'd", "it'll", "it's", "let's", "shouldn't", "t's", "that's", "there's", "they'd", "they'll", "they're",
"they've", "wasn't", "we'd", "we'll", "we're", "we've", "weren't", "what's", "where's", " who's","won't", "wouldn't",
"you'd", "you'll", "you're", "you've", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "zero",
"man", "woman", "wife", "husband", "girlfriend", "boyfriend", "child", "kids", "baby", "toddler", "children", "infant",
"table", "chair", "next", "after", "later", 'from', 'place', 're', 'gmail', 'com','ve','go','s','get','it,','it.',
'would','like','one','though'])
```

```
####### Data Load ########
Data = pd.read_csv('/Users/yash/Downloads/MasterClass/result.csv')

Data.isna().sum()

######### Removing rows with NA in DataSet #########
Data = Data[Data['ReviewerfriendCount'].notna()]
Data = Data[Data['RestaurantRating'].notna()]
Data = Data[Data['reviewContent'].notna()]
Data = Data[Data['reviewID'].notna()]

Data['flagged']=Data['flagged'].replace(['NR'], 'N')
Data['flagged']=Data['flagged'].replace(['YR'], 'Y')
######### Combining Data frame of fake and real reviews #######
FinalData = pd.concat([RealReviews, FakeReviews])


#################### EDA #####################
print("")
print("-------------------------------------------------------------------------------")
print("Status from Basic Observations")
print("-------------------------------------------------------------------------------")
print("")
print("There are {} Observations and {} Features in the Initial Dataframe. \n".format(FinalData.shape[0],
FinalData.shape[1]))
print("There are {} Unique Users in the Dataframe. \n".format(len(FinalData['reviewerID'].unique())))
print("There are {} Unique Restaurants in the Dataframe. \n".format(len(FinalData['restaurantID'].unique())))
print("There are {} Lemmatized Terms from the Reviews Column. \n".format(sum(bigdata.lenReview)))
print("The Longest Review Comprises {} Lemmatized Words. \n".format(max(bigdata.lenReview)))
print("The Shortest Review Comprises {} Lemmatized Words. \n".format(min(bigdata.lenReview)))
print("Number of Real(Flagged as N) and Fake Reviews(Flagged as Y):")
print(FinalData.flagged.value_counts(), "\n")
print("-------------------------------------------------------------------------------")
fig, ax = plt.subplots(figsize=(4, 4))
patches, texts, pcts = ax.pie(pd.Series(FinalData.flagged).value_counts(), labels=FinalData.flagged.unique(),
                    autopct='%.1f%%', wedgeprops={'linewidth': 2.0, 'edgecolor': 'white'},
                    textprops={'size': 'large'}, shadow=True, colors=['cyan', 'red'])
plt.setp(pcts, color='white', fontweight='bold')
ax.set_title('Proportion of Real and Fake Reviews', fontsize=13)
plt.tight_layout()


###### Putting rating into 3 categories and saving it into new column ####
FinalData.loc[FinalData['rating'] == 3,'score'] = 'Neutral'
FinalData.loc[FinalData['rating'] < 3,'score'] = 'Negative'
FinalData.loc[FinalData['rating'] > 3,'score'] = 'Positive'

###### Creating 3 Dataframes to be used in Word Cloud ########
Neutral = FinalData.loc[FinalData['score'] == 'Neutral']
Positive = FinalData.loc[FinalData['score'] == 'Positive']
Negative = FinalData.loc[FinalData['score'] == 'Negative']

##### Distribution of Ratings given by user ########
sns.countplot(x=FinalData['rating'], data = FinalData)
plt.ylabel('Number of Reviews')
plt.xlabel('Ratings given by Reviewer')
plt.title("Distribution of Ratings given by Reviewer")
plt.show()

###### Reviews by ratings and classification #####
sns.set_palette("magma_r")
barGraph = sns.countplot(data = bigdata, x = 'flag', hue = 'rating')
barGraph.set_title('Number of Reviews by Rating and Classification')
barGraph.set_xlabel('Review Classification')
barGraph.set_ylabel('Number of Reviews')
barGraph.legend(title = 'Rating')
plt.show()


### Distribution of Restaurant overall ratings ####
sns.countplot(x=FinalData['RestaurantRating'], data = FinalData)
plt.ylabel('Number of Reviews')
plt.xlabel('Restaurant Ratings')
plt.title("Distribution of Restaurant Overall Ratings")
plt.show()

#### Comparing length of reviews for fake and real ####
plt.figure(figsize=(8,6))
sns.boxplot(x="flag", y="lenReview", data=bigdata,
            palette="magma", showfliers = False)
plt.title('Length of Reviews by Classification')
plt.xlabel("Review Classification")
plt.ylabel("Length of Review")

#### Generate a word cloud image for reviews with Neutral Ratings ####
text = ''.join(Neutral['reviewContent']).lower()
cloud = WordCloud(stopwords = stop_words).generate(text)
plt.figure(figsize=[30,30])
plt.imshow(cloud,interpolation='bilinear')
plt.axis('off')
plt.show()

###### Generate a word cloud image for reviews with Positive Ratings #####
text = ''.join(Positive['reviewContent']).lower()
cloud = WordCloud(stopwords = stop_words).generate(text)
plt.figure(figsize=[30,30])
plt.imshow(cloud,interpolation='bilinear')
plt.axis('off')
plt.show()

###### Generate a word cloud image for reviews with Negative Ratings #####
text = ''.join(Negative['reviewContent']).lower()
cloud = WordCloud(stopwords = stop_words).generate(text)
plt.figure(figsize=[30,30])
plt.imshow(cloud,interpolation='bilinear')
plt.axis('off')
plt.show()

####Generate a word cloud image for reviews with flagged as Y ####
NeutralText = ''.join(Neutral['reviewContent']).lower()
mask = np.array(Image.open("../karan/Downloads/Fake.png"))
```

```
wordcloud = WordCloud(stopwords=stop_words, background_color="white", mode="RGB", max_words=100,
mask=mask, random_state=1,
                    contour_width=2, contour_color='pink').generate(NeutralText)
plt.figure(figsize=(8, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.tight_layout(pad=0)
plt.axis('off')
plt.show()




#### Generate a word cloud image for reviews with flagged as N ####
NeutralText = ''.join(Neutral['reviewContent']).lower()
mask = np.array(Image.open("../karan/Downloads/Real.png"))

wordcloud = WordCloud(stopwords=stop_words, background_color="white", mode="RGB", max_words=100,
mask=mask, random_state=1,
                    contour_width=2, contour_color='pink').generate(NeutralText)
plt.figure(figsize=(8, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.tight_layout(pad=0)
plt.axis('off')
plt.show()

######## Count of fake and real reviews and percentage breakdown #######
Data.flagged.value_counts()
Data.flagged.value_counts()/len(Data)


############### Taking a subset of complete dataset ###########
RealReviews = Data[Data['flagged'] == 'N']
RealReviews = RealReviews.sample(frac= 0.145,random_state = 8)

FakeReviews = Data[Data['flagged'] == 'Y']
FakeReviews = FakeReviews.sample(frac = 0.14,random_state = 8)

######### Combining Data frame of fake and real reviews #######
######### Verifying distribution of real and fake reviews ########
FinalData = pd.concat([RealReviews, FakeReviews])
FinalData.flagged.value_counts()/len(FinalData)




################ Data Cleaning and Pre-processing ################
FinalData['reviewContent'] = FinalData['reviewContent'].map(lambda x: re.sub('[:,\.!?-]', ' ', x))
FinalData['reviewContent'] = FinalData['reviewContent'].map(lambda x: re.sub('\s+', ' ', x))
FinalData['reviewContent'] = FinalData['reviewContent'].map(lambda x: re.sub("\'" , " ", x))
FinalData['reviewContent'] = FinalData['reviewContent'].map(lambda x: re.sub('\S*@\S*\s?' , '', x))
FinalData['reviewContent'] = FinalData['reviewContent'].map(lambda x: re.sub('([()])' , '', x))
FinalData['reviewContent'] = FinalData['reviewContent'].map(lambda x: re.sub('\\*' , '', x))

######### creating bigram measures #########
```

```
 measure_bigram = nltk.collocations.BigramAssocMeasures()
filter_review = nltk.collocations.BigramCollocationFinder.from_documents([comment.split() for comment in
FinalData["reviewContent"]])
filter_review.apply_freq_filter(50)
bigram_scores = finder.score_ngrams(measure_bigram.pmi)

measure_trigram = nltk.collocations.TrigramAssocMeasures()
filter_review = nltk.collocations.TrigramCollocationFinder.from_documents([comment.split() for comment in
FinalData["reviewContent"]])
filter_review.apply_freq_filter(50)
trigram_scores = finder.score_ngrams(measure_trigram.pmi)



pmibigram = pd.DataFrame(bigram_scores)
pmibigram.columns = ['bigram', 'pmi']
pmibigram.sort_values(by='pmi', axis = 0, ascending = False, inplace = True)

pmitrigram = pd.DataFrame(trigram_scores)
pmitrigram.columns = ['trigram', 'pmi']
pmitrigram.sort_values(by='pmi', axis = 0, ascending = False, inplace = True)


##### creating function to apply filter on bigram to consider Noun only #####
def filter_bigram(bigram):
    tagged = nltk.pos_tag(bigram)
    if tagged[0][1] not in ['JJ', 'NN'] and tagged[1][1] not in ['NN']:
        return False
    if bigram[0] in stop_words or bigram[1] in stop_words:
        return False
    if 'n' in bigram or 't' in bigram:
        return False
    if 'PRON' in bigram:
        return False
    return True

def filter_trigram(trigram):
    tagged = nltk.pos_tag(trigram)
    if tagged[0][1] not in ['JJ', 'NN'] and tagged[1][1] not in ['JJ','NN']:
        return False
    if trigram[0] in stop_words or trigram[-1] in stop_words or trigram[1] in stop_words:
        return False
    if 'n' in trigram or 't' in trigram:
         return False
    if 'PRON' in trigram:
        return False
    return True

##### selecting top 500 ngrams in this case ranked by PMI that have noun like structures #####
import nltk
nltk.download('averaged_perceptron_tagger')
latest_bigram = bigram_pmi[bigram_pmi.apply(lambda bigram:\
                                        filter_bigram(bigram['bigram'])\
                                        and bigram.pmi > 5, axis = 1)][:500]

latest_trigram = trigram_pmi[trigram_pmi.apply(lambda trigram: \
                                          filter_trigram(trigram['trigram'])\
                                          and trigram.pmi > 5, axis = 1)][:500]


##### selecting bigrams and trigrams having minimum length of more than 2 characters ########

bigrams = [' '.join(x) for x in latest_bigram.bigram.values if len(x[0]) > 2 or len(x[1]) > 2]
trigrams = [' '.join(x) for x in latest_trigram.trigram.values if len(x[0]) > 2 or len(x[1]) > 2 and len(x[2]) > 2]


###### Join both bigram and trigram #####
def ngram(x):
   for gram in trigrams:
      x = x.replace(gram, '_'.join(gram.split()))
   for gram in bigrams:
      x = x.replace(gram, '_'.join(gram.split()))
   return x

reviews_ngrams = FinalData.copy()

reviews_ngrams.reviewContent = reviews_ngrams.reviewContent.map(lambda x: ngram(x))

#### tokenization of reviews and remove words with less than 2 characters #####
reviews_ngrams = reviews_ngrams.reviewContent.map(lambda x: [word for word in x.split()\
                                       if word not in stop_words\
                                              and len(word) > 2])
reviews_ngrams.head()

##### nouns only ######
def noun(x):
   pos_comment = nltk.pos_tag(x)
   filtered = [word[0] for word in pos_comment if word[1] in ['NN']]
   return filtered

final_reviews = reviews_ngrams.map(noun)

 ########## Create Dictionary ##########
id2word = corpora.Dictionary(final_reviews)
type(corpus)

########## Create Corpus ############
texts = final_reviews

######## Term Document Frequency #######
corpus = [id2word.doc2bow(text) for text in texts]

######### LDA implementation ############
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=5,
                                       chunksize=1000,
                                       passes=50,
                                       iterations = 50,
                                       alpha='auto',
                              per_word_topics=True
                              )


pprint(lda_model.print_topics())

###### Find Perplexity score #####
print('\nPerplexity: ', lda_model.log_perplexity(corpus))

##### Find coherence score #####
coherence_model_lda = CoherenceModel(model=lda_model, texts=final_reviews, dictionary=id2word,
coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)

########### Finding best number of topics of lda results ################
def compute_coherence_values(dictionary, corpus, texts, limit=20, start=5, step=3):
  coherence_values = []
  model_list = []
  for num_topics in range(start, limit, step):
    model = gensim.models.ldamodel.LdaModel(corpus=corpus, num_topics=num_topics, id2word=id2word)
    model_list.append(model)
    coherencemodel = CoherenceModel(model=model, texts=final_reviews, dictionary=id2word, coherence='c_v')
    coherence_values.append(coherencemodel.get_coherence())

  return model_list, coherence_values

model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus,
texts=final_reviews, start=5, limit=20,step=3)




########### Coherence graph ############
limit=20; start=5; step=3;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()


################ Using Mallet ######################
mallet_path = '/Users/karan/Downloads/mallet-2.0.8/bin/mallet' # update this path
ldamallet = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=8, id2word=id2word)

###### Find Perplexity ########
print('\nPerplexity: ', lda_model.log_perplexity(corpus))

coherence_model_ldamallet = CoherenceModel(model=ldamallet, texts=final_reviews, dictionary=id2word,
coherence='c_v')
coherence_ldamallet = coherence_model_ldamallet.get_coherence()
print('\nCoherence Score: ', coherence_ldamallet)
```

```
########### Finding best number of topics of lda mallet model  ################

def compute_coherence_values(dictionary, corpus, texts, limit=20, start=2, step=3):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.wrappers.LdaMallet(mallet_path,corpus=corpus, num_topics=num_topics,
id2word=id2word)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=final_reviews, dictionary=id2word, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values

model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus,
texts=final_reviews, start=2, limit=20,step=3)



######## Coherence graph   ##########
limit=20; start=2; step=3;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()


######### Visuallization of model results ###########
pyLDAvis.enable_notebook()
model = gensim.models.wrappers.ldamallet.malletmodel2ldamodel(ldamallet)
vis = pyLDAvis.gensim_models.prepare(model, corpus, id2word)
pyLDAvis.save_html(vis,'/Users/yash/Downloads/MasterClass14_malllet' + '.html')

pprint(model.print_topics())


######### creating a vector to store topic probabilities ########
train_vecs = []
for index,value in final_reviews.items():
    bow = id2word.doc2bow(value)
    x = model.get_document_topics(bow,minimum_probability=0)
    train_vecs.append(x)


########## storing the vector into a dataframe #########
value = pd.DataFrame(train_vecs)

########### renaming columns in the dataframe and removing comma and topic number from values #######
value = value.rename(columns={value.columns[0]: 'Topic0'})
value = value.rename(columns={value.columns[1]: 'Topic1'})
value = value.rename(columns={value.columns[2]: 'Topic2'})
value = value.rename(columns={value.columns[3]: 'Topic3'})
value = value.rename(columns={value.columns[4]: 'Topic4'})
value = value.rename(columns={value.columns[5]: 'Topic5'})
value = value.rename(columns={value.columns[6]: 'Topic6'})
value = value.rename(columns={value.columns[7]: 'Topic7'})

value['Topic0'] = value['Topic0'].astype('str')
value['Topic1'] = value['Topic1'].astype('str')
value['Topic2'] = value['Topic2'].astype('str')
value['Topic3'] = value['Topic3'].astype('str')
value['Topic4'] = value['Topic4'].astype('str')
value['Topic5'] = value['Topic5'].astype('str')
value['Topic6'] = value['Topic6'].astype('str')
value['Topic7'] = value['Topic7'].astype('str')

value['Topic0'] = value['Topic0'].str.strip('(0,)')
value['Topic1'] = value['Topic1'].str.strip('(1,)')
value['Topic2'] = value['Topic2'].str.strip('(2,)')
value['Topic3'] = value['Topic3'].str.strip('(3,)')
value['Topic4'] = value['Topic4'].str.strip('(4,)')
value['Topic5'] = value['Topic5'].str.strip('(5,)')
value['Topic6'] = value['Topic6'].str.strip('(6,)')
value['Topic7'] = value['Topic7'].str.strip('(7,)')

value["Topic0"] = pd.to_numeric(value["Topic0"], downcast="float")
value["Topic1"] = pd.to_numeric(value["Topic1"], downcast="float")
value["Topic2"] = pd.to_numeric(value["Topic2"], downcast="float")
value["Topic3"] = pd.to_numeric(value["Topic3"], downcast="float")
value["Topic4"] = pd.to_numeric(value["Topic4"], downcast="float")
value["Topic5"] = pd.to_numeric(value["Topic5"], downcast="float")
value["Topic6"] = pd.to_numeric(value["Topic6"], downcast="float")
value["Topic7"] = pd.to_numeric(value["Topic7"], downcast="float")



##### resetting index and merging dataframes of probability and rest of the variables #####

FinalData.reset_index(drop=True, inplace=True)
value.reset_index(drop=True, inplace=True)

bigdata = pd.concat( [FinalData, value], axis=1)

bigdata.isna().sum()

bigdata['flag'] = np.where(bigdata['flagged']== 'N', 0,1)


bigdata['lenReview'] = bigdata['reviewContent'].apply(lambda x: len(x)-x.count(' '))

##### Calculate Correlation Matrix #####
sns.set_style(style = 'white')
plt.figure(figsize=(15, 10))
corrMask = np.triu(np.ones_like(bigdata.corr(), dtype=np.bool))
heatMap = sns.heatmap(bigdata.corr(), annot=True, mask=corrMask, cmap='magma')
heatMap.set_title('Correlation Heatmap of Features in the Final Dataframe', fontdict={'fontsize':13})


#### Dividing variables as dependent variable and response variable ####
```

```
x_features = bigdata.iloc[:,[4,5,10,11,12,13,14,15,16,17,18,19,20,22]]
print(x_features)
y_response = bigdata['flag']
print (y_response)

#### train and test split ####
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_features, y_response,
                                        test_size = 0.3, stratify = y_response,random_state=1)


#### Feature scaling ######
MMS= MinMaxScaler(feature_range = (0, 1))
X_train[[
"rating","usefulCount","ReviewerfriendCount","ReviewerReviewCount","RestaurantRating","lenReview"]]        =
MMS.fit_transform(X_train[[
"rating","usefulCount","ReviewerfriendCount","ReviewerReviewCount","RestaurantRating","lenReview"]])

X_test[[        "rating","usefulCount","ReviewerfriendCount","ReviewerReviewCount","c","lenReview"]]        =
MMS.transform(X_test[[
"rating","usefulCount","ReviewerfriendCount","ReviewerReviewCount","RestaurantRating","lenReview"]])


##### run logistic regression #####
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_logreg = lr.predict(X_test)
print("Test Predictions:{}".format(y_logreg))
accuracy_score(y_test, y_logreg)

##### KNN Algorithm ######
knn = KNeighborsClassifier(n_neighbors=5,leaf_size=15)
knn.fit(X_train, y_train)
y_knn = knn.predict(X_test)
print("Test Predictions:{}".format(y_knn))
accuracy_score(y_test, y_knn)

#### Grid Search CV for KNN #####
clf = KNeighborsClassifier(n_jobs= -1)

param_grid = {
    'n_neighbors': range(1,21,2),
    'leaf_size' : [2,6,10,15,20],
}

scorers = {
    'recall_score': make_scorer(recall_score, average = 'weighted')
}

def grid_search_wrapper(refit_score='precision_score'):
    """
    fits a GridSearchCV classifier using refit_score for optimization
    prints classifier performance metrics
    """
    skf = StratifiedKFold(n_splits=5)
    grid_search = GridSearchCV(clf, param_grid, scoring=scorers, refit=refit_score,
                          cv=skf, return_train_score=True, n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # make the predictions
    y_knn = grid_search.predict(X_test)

    print('Best params for {}'.format(refit_score))
    print(grid_search.best_params_)

    return grid_search

grid_search_clf = grid_search_wrapper(refit_score='recall_score')

# Random Forest
rf = RandomForestClassifier(n_estimators=200, criterion='entropy', min_samples_split = 2, max_depth=25,
max_features=None)
rf.fit(X_train, y_train)
y_rf = rf.predict(X_test)
print("Test Predictions:{}".format(y_rf))
accuracy_score(y_test, y_rf)

##### Grid Search CV for Random Forest #####
clf = RandomForestClassifier(n_jobs=-1)

param_grid = {
    'min_samples_split': [2],
    'n_estimators' : [200,400],
    'criterion' : ['entropy'],
    'max_depth' : [13,15,17,20,25,30],
    'max_features' : [None],
    'class_weight' : ['balanced_subsample']
}

scorers = {

    'recall_score': make_scorer(recall_score, average = 'weighted')
}


def grid_search_wrapper(refit_score='precision_score'):
    """
    fits a GridSearchCV classifier using refit_score for optimization
    prints classifier performance metrics
    """
    skf = StratifiedKFold(n_splits=5)
    grid_search = GridSearchCV(clf, param_grid, scoring=scorers, refit=refit_score,
                          cv=skf, return_train_score=True, n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # make the predictions
    y_rf = grid_search.predict(X_test)

    print('Best params for {}'.format(refit_score))
    print(grid_search.best_params_)
    return grid_search

grid_search_clf = grid_search_wrapper(refit_score='recall_score')



##### run SVM #####
svc = SVC(C=10, gamma = 1)
svc.fit(X_train, y_train)
y_svc = svc.predict(X_test)
print("Test Predictions:{}".format(y_svc))
accuracy_score(y_test, y_svc)










###### Confusion Matrix ######
rf_matrix = confusion_matrix(y_test, y_logreg)
disp = ConfusionMatrixDisplay(confusion_matrix = rf_matrix,
                display_labels=['0','1'])
disp.plot(cmap= 'Blues')
plt.title('Confusion Matrix of Logistic Regression')


####### Computing Precision and Recall scores ######
matrix=classification_report(y_test, y_logreg)
print(matrix)


####### Recall and precision curve #######
yhat = lr.predict_proba(X_test)
pos_probs = yhat[:, 1]
no_skill = len(y_response[y_response==0]) / len(y_response)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
precision, recall, _ = precision_recall_curve(y_test, pos_probs)
pyplot.plot(recall, precision, marker='.', label='Logistic')
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()
pyplot.show()
```