Queen Mary University of London

CLASSIFYING FAKE AND REAL YELP REVIEWS USING TOPIC MODELLING

BUSM131 - Masterclass in Business Analytics
2021/22

SWASTIK VITTHAL CHAVAN
2000933614

Arjun Mukherjee, Vivek Venkataraman, Bing Liu, and Natalie Glance. What Yelp Fake Review Filter Might Be Doing. Proceedings of The International AAAI Conference on Weblogs and Social Media (ICWSM-2013), July 8-10, 2013, Boston, USA.

Table of Contents

ABSTRACT

Customers use Yelp online reviews as their primary source of information when making purchasing decisions. However, online reviews are not always truthful as fake reviews are created for commercial purposes in order to mislead buyers. For the past decade, Yelp's algorithm has been blocking out reviews. Based on the research it reveals that the yelp dataset is categorising the filtered data is incorrect. The project consists of a topic modelling approach for supervised categorization of real and fake reviews to figure out what Yelp is trying to do by filtering its reviews. We address this issue by developing a classifier that takes the review text as input, as well as the reviewer's general information, and evaluates whether the review is reliable. By incorporating different techniques into classifying models, this paper aims to improve the performance of fake review classifiers. The output is to detect fake reviews based on both feature extraction and deep-learning methods. We use the distribution of the probability of each topic to determine the learning algorithms, including logistic regression, decision trees, and random forests. According to the results, the random forest performs the best, with a recall of 0.84 for classifying real reviews and an accuracy of 70 %. By combining dictionary-based information, document term matrices, optimal topics, and non-textual features of reviews and reviewers, we construct machine learning models to identify fake reviews. Surprisingly, the behavioural features perform well, but the linguistic features do not have as much impact. In this study, we compare three supervised machine learning algorithms which are Logistic Regression, Decision Tree, and Random Forest. Based on our experiments, the Random Forest algorithm outperformed other algorithms.

1. INTRODUCTION:

People consider online reviews when making purchases and decisions about their businesses. Yelp is one such website that is designed to offer information or receive recommendations through customer reviews about the quality of different types of amenities through which consumers may decide whether to opt for it. It is also a social network as customers can participate in an ongoing conversation about those amenities that allow them to designate their friendship in addition to a business search or a rating website. It is a platform that helps customers find places or companies such as restaurants, hotels, and more using reviews on the site. Yelp understands the potential threat can cause inaccurate information to the customers as the website generates a lot of traffic website with almost 800 million monthly visits. Due to increased popularity in online reviews, fake reviews or deceptive reviews are used to influence reader opinions. As positive reviews are in a direct relationship of popularity and revenues for businesses which, unfortunately, provides huge incentives for imposters to write fake reviews in order to promote or criticize certain items or services. Yelp established a policy to address this issue and used an algorithm to automatically filter all reviews which can be considered as harmful.

Machine learning algorithms are beneficial to increase the accuracy of classifying fake reviews as classification techniques learn from the data and distinguish between real and fake reviews. The Yelp algorithm was also used to detect false restaurant reviews and it was discovered that the algorithms worked for behavioural but not so well for linguistic aspects (Mukherjee, et al., 2013). (Luca & Zervas, 2016). SVM algorithms have also been found quite beneficial for classification, outperforming other typical data mining techniques in text classification. Various other factors such as useful counts of a review can also be helpful to identity if the reviews are real or fake. (Wahyuni & Djunaidy, 2016). In experimentation, it was also discovered that the accuracy of the classification algorithms with both n-gram and psychological deception aspects improves by about 90%. (Ott, et al., 2011). Furthermore, standard n-gram text categorization systems outperform human judges when it comes to detecting negative misleading opinion spam. (Ott, et al., 2013).

The main goal of this research is to use various machine learning algorithms using natural language processing (NLP) such as topic modelling to develop a model that determines whether a review is true or fake. The supervised algorithms used to consist of linguistic features such as bigrams or trigrams but instead of using them directly to fit a model we use Latent Dirichlet Allocation (LDA) to obtain ideal topics based on coherence scores. Each topic has a probability distribution over the words which are calculated over each review which are further used to input in the learning models. Various other features from reviews such as useful count, funny count, are used in the classifier model. The experiment consists of supervised classification algorithms such as Logistic Regression, SVM, and XG Boost to compare their performance which can be further used to detect fake reviews using the optimal model.

2. BUSINESS PROBLEM:

This topic has received a lot of attention on review sites, where the spread of misinformation in the form of opinion spam, as well as the negative effects that it brings, is especially destructive to both businesses and users. Customers are presented with fake statistics which encourage products or services of inferior quality and have the potential to cause harm. Companies receive fake positive ratings to boost sales, or they get negative reviews on competitors to drive them down. Considering the Yelp dataset, the users suffer from two problems related to fake reviews.

- Fake reviews can appear to be an amazing deal that can manipulate the user's decision based on the positive reviews of the product which shows a lot of users. Thus, the user opts for the service or product and is dissatisfied with the result.
- Another problem stated in the customer's trust in the Yelp platform will start declining as fake reviews are not being filtered out.

Analysis of the Existing System:

The results of Yelp's filtering algorithm predict if the review is real or fake and flags the suspicious reviews to remove them from the main page. The algorithm is not disclosed by Yelp, but the outputs are available to analyse the problem associated with it. Only 16% of restaurant fake reviews are filtered by Yelp which shows that the algorithm is not perfect. Thus, there are both filtered reviews that are considered not fake and fake reviews that are not filtered. The fake reviews can be higher or lower than 16% of filtered reviews and the reviews which are not fake are also being filtered. Thus, restaurants which genuine positive reviews are also filtered. Similarly, the restaurants with no filtered reviews might have faked reviews for profits. Such misclassification of results can cause fraud in the review systems.

Business Proposal:

Yelp filters fake reviews to keep the authenticity of the reviews based on behavioural features which are not accurate. In this problem, we classify fake and real reviews using linguistic features such as bi-grams and trigrams which increases the accuracy of the algorithms. The reviews can be effectively measured by using topic modelling and probability distribution of each topic in the review. This result can contribute to the growing challenges of user-generated quality content which in turn helps ethical practices in the restaurants due to changing competition and reputation.
There is a problem with the trustworthiness of online evaluations because around 20% of Yelp reviews are allegedly manufactured by human writers paid by the businesses they evaluate. Because the problem has gotten so bad, Yelp.com organised a sting operation to expose businesses that buy bogus ratings. The aim is to provide an algorithm that can detect false reviews and offer users trustworthy information about them. This model takes into account the content of a review as well as the information provided by the reviewer. The validity of the review is then determined using neural networks.

Grey Literature:

The University of Illinois at Chicago's Arjun Mukherjee and Bing Liu cooperated with Google's Natalie Glance, who provided a Google Faculty Award to help fund the research. The research focuses on the GSRank algorithm, which considers relationships between groups, individual reviewers, and the products evaluated. The software detects the review threads that are attempting to control sentiment and classify the document's spam, allowing it to be ranked and dealt with appropriately. GSRank surpassed all other algorithms in use, demonstrating that spam reviewers can be identified. (Brown, 2012)

A was research done by Davide Proserpio, Brett Hollenbeck, & Sherry He, through a private Facebook group where sellers used these groups to hire people to buy their products and add positive reviews or 5-star ratings. The companies would then pay back the money using PayPal or any other online transaction mode or sometimes even give them a commission. These groups would eventually vanish, and new ones were created. They worked with the UCLA undergraduates to sneak into the market and gather relevant information about the products and sellers who were asking for false reviews. On the other hand, they also gathered the data from Amazon which included ratings, reviews, ranking, etc. This helped them to quantify the success of these reviews when sellers started and stopped soliciting fake reviews and compare that activity to their product's sales statistics. (Proserpio, et al., 2020)

To construct a gold-standard fake reviews dataset, the authors of (Ott, et al., 2011) used crowdsourcing via Amazon Mechanical Turk (AMT Humans can't detect if a review is fake just by reading it, according to the study, indicating an at-chance probability. It combined psycholinguistic characteristics with the most common POS tags in the review text using an n-gram model. As a result of the tests, words associated with inventive writing were revealed. While the classifier performed well on the AMT dataset, it's possible that spammers may simply avoid using it if they learn about it. Models evaluated just against the artificially constructed AMT dataset do not generalise to real-world conditions, as proved by (Mukherjee, et al., 2013). This study used Yelp reviews to rebuild and evaluate the model of (Ott, et al., 2011), assuming that Yelp has mastered its fraud detection systems over its decade-long history. Surprisingly, instead of the 90 percent accuracy stated on AMT data, they found just 68 % accuracy when they assessed Ott's model using Yelp data.

3. DATA, EDA, AND METHODS:

3.1 Data Description:

The following Yelp Dataset consists of 7,88,471 observations and 10 columns with 3,26,981 Fake reviews and 4,61,490 Real reviews. This dataset was provided by Bing Liu, author of the paper (Mukherjee, et al., 2013). There are 2 databases one for hotels and the other for the restaurant. Each of the databases consists of having 3 tables named Review, Reviewer, and Restaurant/Hotel. We will be using the Restaurant Database only where the review table is the main table. There are primary keys such as restaurant, reviewerID which are used to join the review table. The attribute "date" contains posted review date. We have considered 1,00,000 subsets of the complete dataset due to computational time with an equal number of fake and real reviews for accurate analysis.
The following variables are used for classification:

Table 1: Data Descriptions

Variable	VariableType	DataType	Description	
reviewContent	Independent	Char	Written reviews by the reviewer	
reviewRating	Independent	Integer	Rating of the review (1 to 5)	
usefulCount	Independent	Integer	Total number of useful votes of the reviewer	
coolCount	Independent	Integer	Total number of cool votes of the reviewer	
funnyCount	Independent	Integer	Total number of funny votes of the reviewer	
reviewCount	Independent	Integer	Total number of written reviews by the reviewer	
restaurantRating	Independent	Integer	Rating of the restaurant (1 - 4.5)	
complimentCount	Independent	Integer	Total number of compliments to the reviewer	
tipCount	Independent	Integer	Total number of tips of the reviewer	
fanCount	Independent	Integer	Total number of fan votes of the reviewer	
Flagged	Dependent	Object	*Fake* *Y- filtered* *YR-unfiltered*	*Real* *N-filtered* *NR-unfiltered*

3.2 DATA PRE-PROCESSING:

Data Cleaning:

In this Yelp dataset, we check for null values after importing and merging the tables with the review table. We check the count and percentage of the missing value from the total dataset. As it is shown below in figure 1, there are 80,203 NaN values present in the complete dataset which is approximately 10%. Dropping the rows or columns that contain null values is the simplest technique to handle this problem. We also remove any unnecessary columns from the main table such as Date and IDs before modelling.

	Count	Percentage
fanCount	80203	10.171966
tipCount	80203	10.171966
complimentCount	80203	10.171966
friendCount	80203	10.171966
reviewCount	7671	0.972896
restaurantRating	7671	0.972896
reviewContent	2	0.000254
reviewID	1	0.000127
date	0	0.000000
flagged	0	0.000000
restaurantID	0	0.000000
funnyCount	0	0.000000
coolCount	0	0.000000
usefulCount	0	0.000000
reviewRating	0	0.000000
reviewerID	0	0.000000

Figure 1:Percentage of missing data

The flagged data consists of 4 categories of filtered and unfiltered data which as ‘Y’, ‘YR’, ‘N’, ‘NR’. “Y” means filtered reviews by Yelp system fake and “N” mean not fake. NR/YR ais unfiltered data in the columns. Thus, we classify N/NR as True and Y/YR are fake reviews and generate an additional column called as given_class.

Figure 2: Text pre-processing steps

The review content needs to be cleaned by performing some NLP pre-processing steps which can be used for analysis. The following steps have been used to transform the unstructured review content:

- Tokenization- The reviews are broken down into words or token by splitting the text into sentences and then sentences into words. At this step we also use the regular expressions such as removing punctuations and lowercasing the words.
- Stop Words Removal – We first initialize a custom stop words list and remove them to reduce the characters and retain important words.

- Lemmatization- This process transforms the words to its root word as the algorithm refers to a dictionary to reduce the words by its meaning.

3.3 EXPLORATORY DATA ANALYSIS:

We have distributed the fake and real reviews by 25% each from the given_class variable which states true or false. Out of the these we consider only 1,00,000 reviews to avoid any skewed data which may end up with a higher accuracy, but it may assume most of the reviews as true. Thus, the distribution of fake and real comes a s 58.5% and 41.5%.

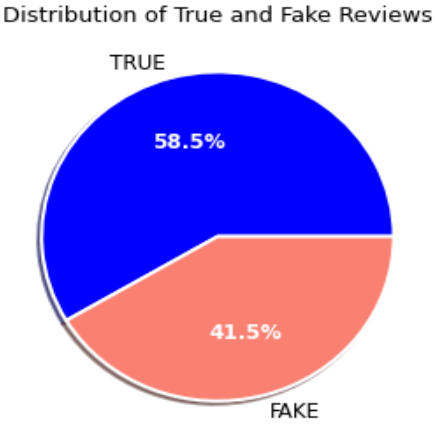


Figure 3:Pie chart - Distribution of Fake and Real Reviews.

After the text pre-processing step, we construct a Word cloud for both Real and fake reviews to check if the cleaning was done appropriately. This also shows us the most frequently used word in the reviews for both real and fake.

Figure 4: Word Cloud for Real Reviews

Figure 5:Word Cloud for Fake Reviews

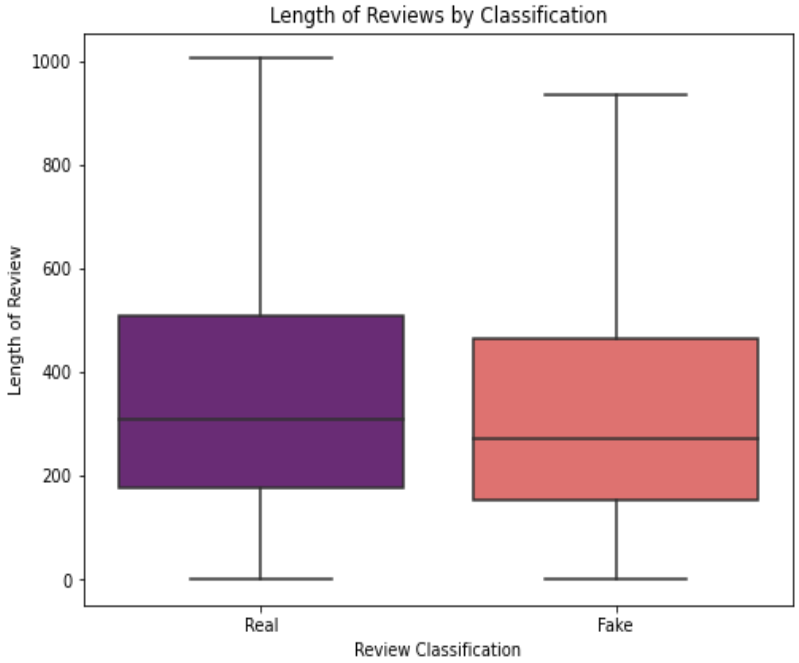


Figure 6: Box Plot - Length of fake and real reviews

The term frequency is how many times the word is occurring in the review. It can be calculated as the length of the review divided by the total number of words in the review. Thus, length of the review is also a factor to classify a review as fake and real. It is clearly visible that there are more positive words with a mean of approx. 300 and fake words is 250 words in the whole review content.

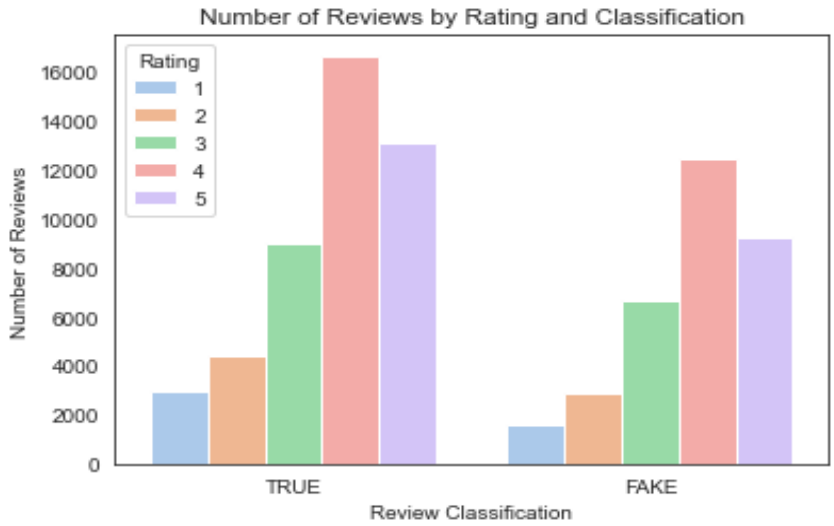


Figure 7: Review Classification- reviewRating vs Number of reviews

The review rating ranges from 1 to 5 and it's clearly visible that for both True and Fake reviews the data is normally distributed which as both the means will be symmetrical.

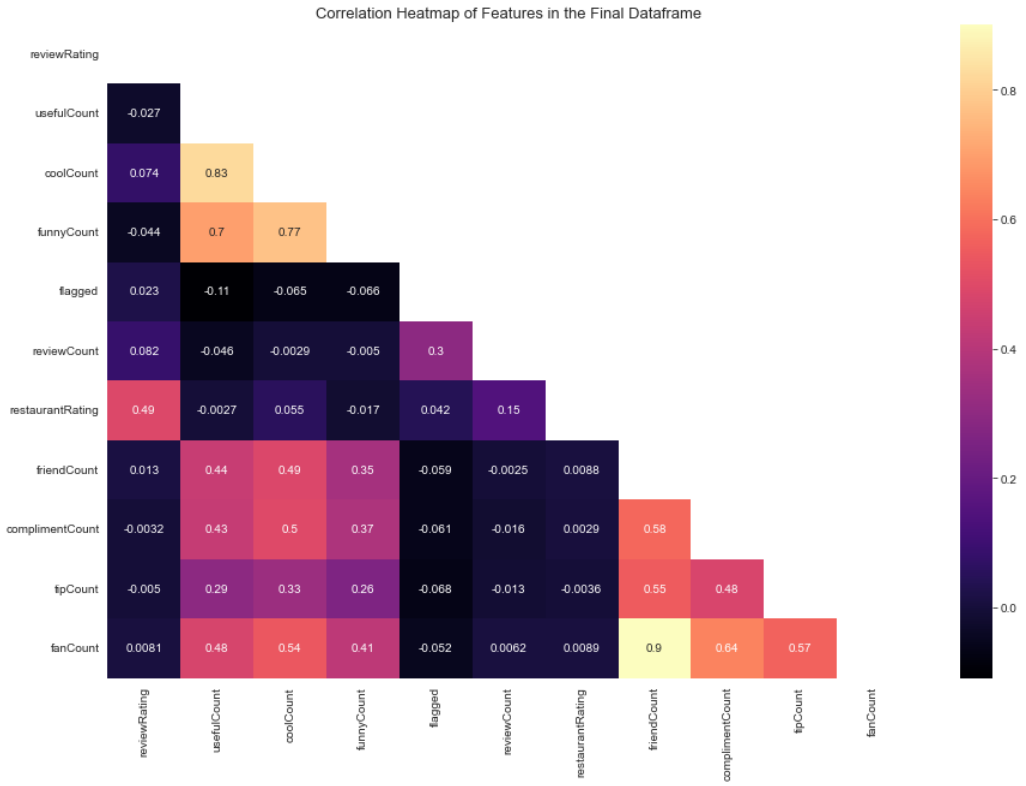


Figure 8: Correlation heatmap of all features

The correlation heatmap shows the important features which can be used for modelling. It can be seen that behavioural features such as friend count, fan count has the highest correlation as the votes of the fans and friends on the review is the highest. The correlation of coolCount, usefulCount also tells us that useful votes are considered as cool as well as coolCount are also considered to be funny. We only use one of these features as they are highly correlated with each other which can be biased for modelling.

3.3 FEATURE EXTRACTION:

This process consists of two parts: Pre-topic modelling and Post-topic modelling features.
Feature extraction is used to extract the original features of lower dimensions to transform into a high dimensional space data. After the pre-processing steps we extract the feature vectors which are necessary to construct two separate dictionaries for bigrams and trigrams. These are setup n-grams identification for genism which shows only one word that occurs 5 times. The frequency of n-grams does not seem to be high, so we exclude it and move further. The bag of words (BoW) represents each review as a numeric vector where each dimension is a specific word from the corpus and the value can be its frequency such as 0 or 1 or any even numbers. It creates a dictionary from these documents as a training set. As the pre processing steps such as tokenizing and lemmatizing, the bag of words represents as a dictionary where the word is the key and value is the number of times it occurs in the corpus. This matrix is termed as Corpus which is a document term matrix that relates to the reviews and through bag of words.

dictionary = gensim.corpora.Dictionary(Clean_data)

texts=clean data
corpus = [dictionary.doc2bow(text) for text in texts]

Data vectors using get document topics model is used to find the probability distribution of each topic. It means that the probabilities of hidden words provide an explicit representation of the entire collection of reviews. Latent Dirichlet Allocation (LDA) model is used to get an approximate model form the topic model where the document is represented which are randomly mixed. This give the topic class where each word belong to a particular topic and each topic has some distribution over the vocabulary. is a generative probabilistic model, from the topic model class, in which the documents may be represented as random mixtures of topics, and each topic may be modelled as a distribution over words/terms present on the dataset (vocabulary). The final model is used to determine the number of topics used to get the probability if each topic which is considered for classification as an input for supervised learning models to determine the reviews as fake or real.

3.5 MACHINE LEARNING MODELS:

In detecting the fake reviews, we used supervised algorithms which are based on feature extraction and user behaviours. The project builds a Latent Dirichlet Allocation (LDA) model to obtain new features. The model consists of training a LDA model to feed into the supervised algorithm for classification. The input data for LDA is the document term matrix and a dictionary. It rearranges the topics within the document to find a good keyword distribution. The task is to the get the topic distribution per review using Dirichlet distribution. I have used a parallelization of the LDA called as LDA multicore which uses CPU cores to speed the training models as the review content takes a lot of time to run. The output is shown in as an interactive designed visualization where the user is able to interact with the number of topics to the corpus of the text using pyLDAvis package. Genism libraries are used to create a unique representation of each text in the document which is further used for mapping.

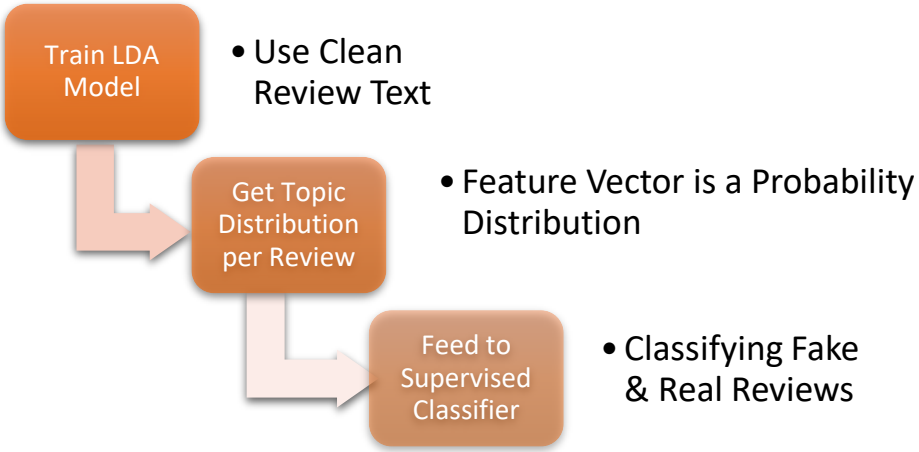


Figure 9: Data flow for models

Then, the topics extracted may be used as input features, once each review can be represented as a vector that indicates the probability distribution of this document over the selected topics. It is assumed that during the process of drafting the text, its generation process, the author exchange by several of these topics, using the words belonging to each of them. That is, the words from different topics are allocated by the result of the Dirichlet distribution sample result, and, by this process, the document is populated. Important to note that documents may have the same topics but still be different because these documents contain different proportions of these topics.

In terms of learning models, the input are the probability distribution of the topics and features from the review tables which are used to apply Logistic Regression, Decision Tree and Random forest models. For hyperparameter tuning in each classification, the dataset is divided into 80 training and 20% testing. The supervised machine learning problems use sklearn.model_selection library for test and train split. The goal is to obtain proper hyperparameters for classifying real or fake reviews. In logistic regression we use the probability of the topics considered as fake (0) or real (1) by max likelihood method. Decision tree algos are built over a binary tree where the values will split the value until a leaf node is reached. The iteration of texts present and the significant weights or vectors are analysed to an optimal one by information gain where it will be used to classify reviews as legitimate or not. Random forest is a bagging technique where it creates a set of decision trees by randomness of the training set. RandomForestClassifier method is used from the sklearn library to use the fit and predict methods. Different models are created, and results are then aggregated.

4. ANALYSIS AND RESULTS:

LDA model:

The LDA multicore model results the number of topics with the weights assigned to that keyword to show the importance. To measure if the model build, we use perplexity and topic coherence scores. Model perplexity represents the model's average level of uncertainty for each text in the review content while topic coherence is used to evaluate the semantic interpretability of the obtained topics. Lower the model perplexity better the model.

```
Perplexity:  -7.748619034504948

Coherence Score:  0.45897811533125504
```

Hyperparameters used in the LDA multicore model:
The model consists of hyperparameters used to optimize training for models which increases the computational time.

- chunksize (int, optional) = 2000, are the number of documents to be used for training and as much the memory can handle it
- passes (int, optional) = 60, are the total number of passes during training through document matrix.
- iterations (int, optional) = 80, total number of iterations through the matrix while referring to the topic distribution
- random_state[(np.randomstate,int)=100, It is useful for reproductibility as it is a seed to generate a randomState object.

We visualize the topics

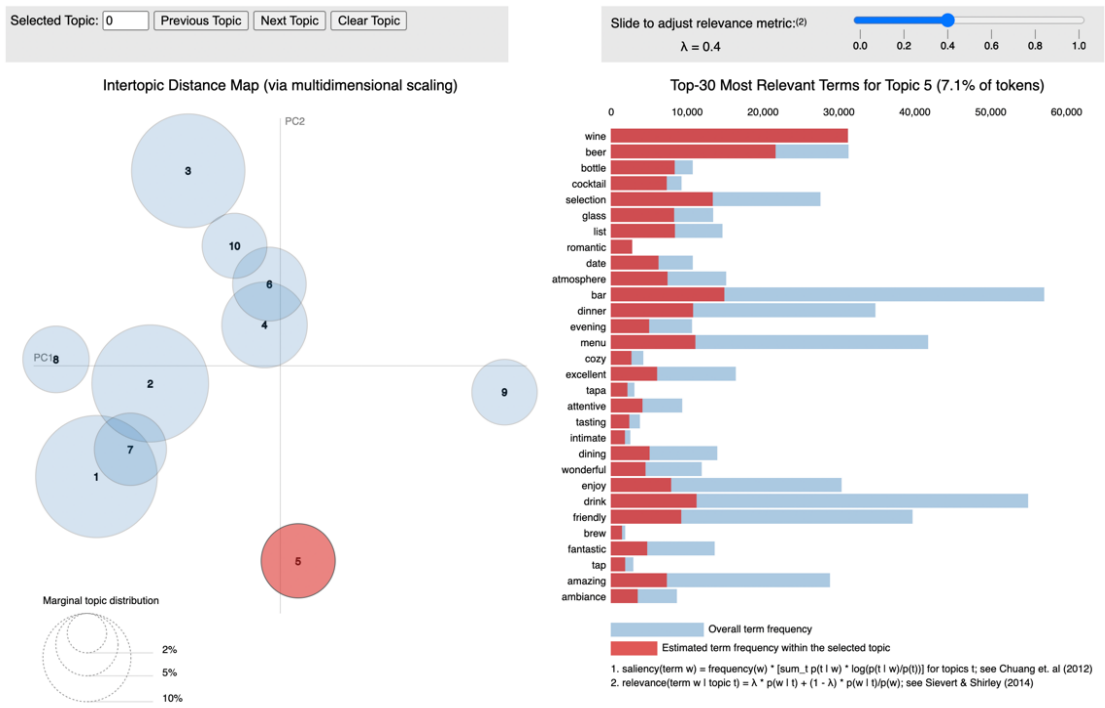


Figure 10: LDA Model

pyLDAvis output consists of four larger bubbles which are the more frequent topics occurred. After applying hyperparameters we can see that there still high number of topics in smaller bubbles are still overlapping as there are still common words which can be filtered. The distance between the bubbles is the relationship between them. When we hover over a topic such as topic 5, we can see that the frequency of these topics is comparatively less. However, decreasing the lambda parameter, increases the ratio of frequency of each word in the topic and the words increase for better interpretability. To find the optimal model we find the coherence values of 20 topics and see that there is a sharp rise on topic 9.

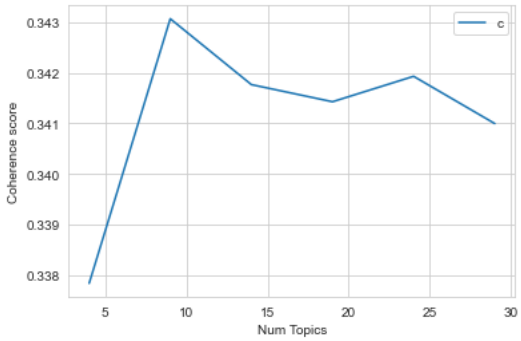


Figure 11: Topic Coherence Scores

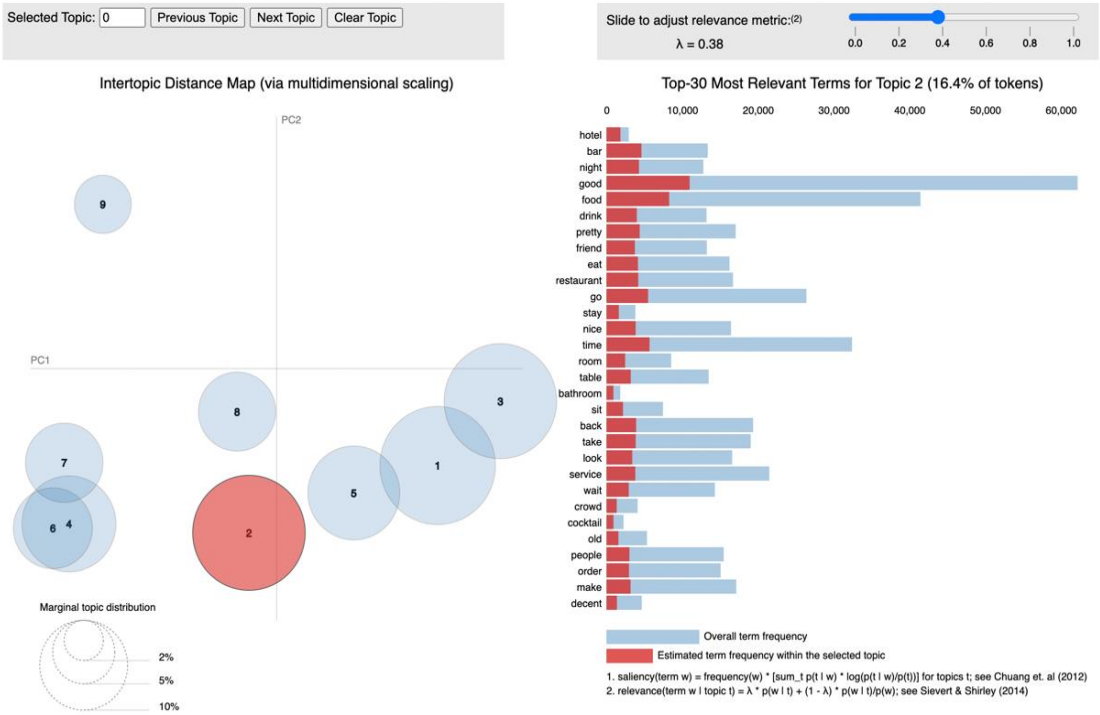


Figure 12:Optimal LDA Model

The optimum model is used to choose the final number of topics based on the highest coherence. Thus, topic 9 gives meaningful insights as later it declines and flattens out. Thus, we choose the final topics as 9 to find the probability distribution of each topic in each review which can be then used for classification.

CLASSIFICATION OF REAL AND FAKE REVIEWS:

It is important to correctly identify the right matrix for the business problem. The metrics used for these modelling is precision, recall, F1 score and accuracy. In classifying the reviews,

- False Positives= Model predicts the review as fake when it's actual real.
- False Negative= Model classifies real review as fake.

According to the hypothesis testing, Type error 1 rate occurs when we reject the null hypothesis which is true. (False Positive) while type 2 error is probability of failing to reject the null hypotheses when it's actually false. As the business problem states to help Yelp website gain the faith of the users, fake reviews are not correctly classified and real reviews are being filtered out which can damage Yelp's reputation. As user's trust declines the interest of using yelp or any services will reduce with time. Thus, we consider Recall as a suitable matrix as we want to correctly classify real reviews which are not fake. This helps both the honest businesses on Yelp as well as the users using the platform for accurate information.

- Precision: TP/(TP+FP) how many actual positive reviews from all anticipated positive cases can the model capture?
- Recall: TP/ (TP+FN) Out of all the positives in the review dataset, how many reviews did we accurately identify as positive?
- F1 Score: The harmonic mean of precision and recall is the F1 score.
- Accuracy: How many reviews did the models properly classify? (TP+TN)/(TP+FP+TN+FN)

Table 2:Comparison of Evaluation Metrics

			Precision	Recall	F1-Score	Accuracy
Classifier	Logistic regression	0	0.60	0.89	0.72	59%
		1	0.55	0.19	0.28	
	Decision Tree	0	0.61	0.71	0.66	57%
		1	0.48	0.37	0.42	
	Random Forest	0	0.61	0.84	0.71	60%
		1	0.55	0.28	0.37	

The results in Table 1 show that Logistic model has a precision of .60 which is improved by using regularization hyperparameters such as penalty=12. In other words, when the model predicts a review true, it is correct 60% of the time. Similarly, the models recall is 0.89 it shows 89% of all real reviews. The accuracy comes to 69%. Similarly, I want to limit the depth to which each tree in my random forest can grow, thus we use hyperparameters such as max_depth and min_sample_split. The decision tree correctly identifies real reviews by 71%. However, Random forest recall score is 84% the accuracy seems to be higher. Thus, we consider Random forest and choose to perform further parameter tuning to increase recall and accuracy.

5. DISCUSSIONS AND CONCLUSION:

The Yelp dataset provided by the author of the paper (Mukherjee, et al., 2013), had precise theoretical analysis based on the linguistic features difference. Filtered Yelp reviews show obvious dishonesty and pretence, implying that filtered reviews are fraudulent. Using behavioural traits improves accuracy even further. These are not likely to be displayed by a real reviewer. Thus, we classify the data using topic modelling which can be depicted through a process flow diagram. The accuracy of classifying real or fake is 60% for Random forest which correctly identifies real reviews by 87%.

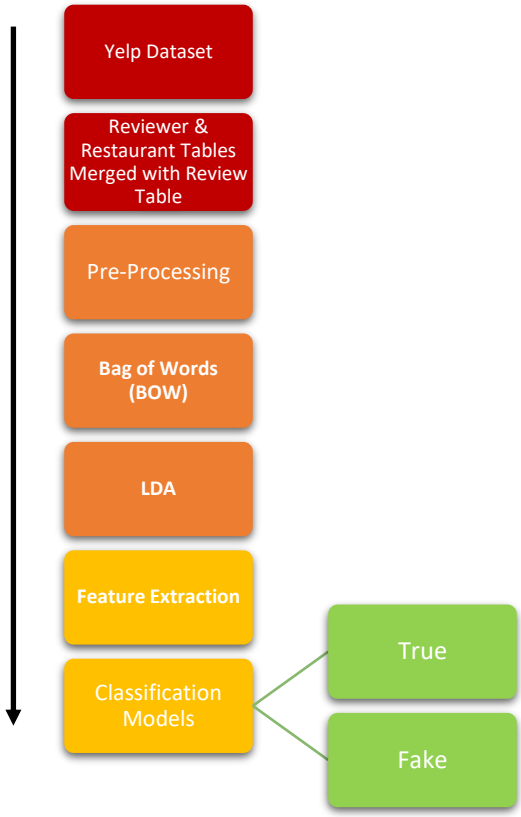


Figure 13: Process Diagram for Classification of Fake and Real reviews.

Limitations: The LDA model takes a lot of time to run due to hyperparameter tuning of passes and iterations. The database consists of approx. 8,00,000 data with 3 tables with various features that can be used for classification such as restaurants who serve alcohol, deliveries, etc. I would also like to explore more of the linguistic features as only one bi gram topic was visible due to low frequency of occurrence. Due to computational time we consider a subset of a data as it takes a lot of time to clean and run topic modelling. However, if we take a large number of dataset running on a high processor GPU, the accuracy will be higher. Text cleaning is usually a game changer, and lemmatization can be improved by employing different enhancing models like Wordnet. Data pre-processing such as stemming and lemmatization processes, it is possible to reduce the dimensionality by reducing the size of particular features by replacing groups of texts with corresponding tokens. Hyperparameters used were limited in the classification techniques, more tuning can lead to better results. Deep learning algorithms such as CNN, Bi-directional LSTM can be implemented to improve the overall performance.

References

1. Brown, E., 2012. *Writing fake online reviews? New Google algorithm will catch you out.* [Online]
   Available at: https://www.zdnet.com/article/writing-fake-online-reviews-new-google-algorithm-will-catch-you-out/
2. Jindal, N. & Liu , B., 2008. *Opinion Spam and Analysis.* s.l., WSDM '08 11.
3. Luca, M. & Zervas , G., 2016. Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud. *Management Science 62(12):3412-3427 . https://doi.org/10.1287/mnsc.2015.2304.*
4. Mukherjee, A., Venkataraman, V., Liu, B. & Glance, N., 2013. Fake Review Detection: Classification and Analysis of Real.
5. Mukherjee, A., Venkataraman, V., Liu, B. & Glance, N., 2013. "What Yelp Fake Review Filter Might Be Doing?". *International AAAI Conference on Web and Social Media.*
6. Ott, M., Cardie, C. & Hancock, J. T., 2013. Negative Deceptive Opinion Spam. *2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,*, p. 497–501.
7. Ott, M., Choi, Y., Cardie, C. & Hancock, J. T., 2011. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. *49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies,*, p. 309–319.
8. Proserpio, D., Hollenbeck, . B. & He, S., 2020. *How Fake Customer Reviews Do — and Don't — Work.* [Online]
   Available at: https://hbr.org/2020/11/how-fake-customer-reviews-do-and-dont-work
9. Wahyuni, E. D. & Djunaidy, A., 2016. FAKE REVIEW DETECTION FROM A PRODUCT REVIEW USING MODIFIED METHOD OF ITERATIVE COMPUTATION FRAMEWORK. *MATEC Web of Conferences, vol. 58, p. 03003.*

Peer Assessment Form
for the Masterclass in Business Analytics (BUS131)

Team Number/Name(s): Team J1 | Sakshi, Sinduja, Swastik, Yash

As a team you should keep a record of your meetings, attendance, and performance/contribution of each of your team members based on the statements below.

The team work requires you to be pro-active, i.e. if you notice issues in your team work, you should first discuss them internally and try to work them out as a team. If you cannot resolve them after having made all possible efforts, please notify the module organisers. In such casesthe module organisers will have a meeting with the team to resolve any issues.

If you cannot reach one of your team members, please copy the module organisers in your emails.

Please fill out and submit below peer assessment form as a team on the end of the year on QMplus. Please append the form to your final essay. As a default all team members receive thesame grade. However, in cases in which team members cannot come to an agreement regardingtheir contributions, each individual within the group should submit a separate peer assessmentform via email to the module organisers. In this case individual grades may be altered. The module organisers reserve the right to alter the marks of individuals within a team in such a way that they reflect the effort of individual members. Your presentation and essay will not bemarked until peer assessment forms have been received.

Please list all your team members, including yourself, by name in the tables below (left column)and then tick (x) the appropriate boxes for each group member.

1. Meeting attendance & punctuality

Team Member	Attended all meetings & arrived on time	Attended less than half of themeetings	Never attended any meetings
Sakshi Shreepal Jain	Yes		
Sinduja Natarajan	Yes		
Swastik Vitthal Chavan	Yes		
Yashovardhan Bhomia	Yes		

2. Contribution to meetings/group discussions

Team Member	Contributed meaningfully to all meetings	Contributed something to less thanhalf of meetings	Never contributed to meetings
Sakshi Shreepal Jain	Yes		
Sinduja Natarajan	Yes		
Swastik Vitthal Chavan	Yes		
Yashovardhan Bhomia	Yes		

Quality of work

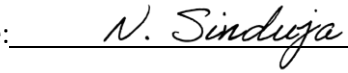
Team Member	Consistently produced high quality work	Work tended to be of average quality	Work tended to be of poor quality	No work produced
Sakshi Shreepal Jain	Yes			
Sinduja Natarajan	Yes			
Swastik Vitthal Chavan	Yes			
Yashovardhan Bhomia	Yes			

3. Overall positive contribution to the team assignment, including discussion, code, analysis and presentation

Team Member	Has contributed significantly	Has at times made contribution	Has only made a small contribution	Has failed to make any contribution
Sakshi Shreepal Jain	Yes			
Sinduja Natarajan	Yes			
Swastik Vitthal Chavan	Yes			
Yashovardhan Bhomia	Yes			

We confirm that all members of our group have reached the above consensus:

Name: Sakshi Shreepal Jain Signature:  Date: 10-Jan-22

Name: Sinduja Natarajan Signature:  Date: 10-Jan-22

Name: Swastik Vitthal Chavan Signature:  Date: 10-Jan-22

Name: Yashovardhan Bhomia Signature:  Date: 10-Jan-22

Appendix:

Python Code:

Libraries used:

```
import pandas as pd
import numpy as np
import seaborn as sns
import re
from pprint import pprint


import nltk; nltk.download('stopwords')


# spacy for data lemmatization
import spacy

# importing Gensim libraries
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# Plotting tools and LDA plot
import pyLDAvis
import pyLDAvis.gensim_models
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from textblob import TextBlob

import tensorflow as tf
# Importing skllearn for classifcation algorithms
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, explained_variance_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import auc, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from keras.models import Sequential
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras import optimizers
from keras import applications
```

```
from keras.models import Model
from keras import backend as K
# from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier



# importing Yelp .csv Datasets for Restaurants, Reviews, and Reviewers
restaurantdf=pd.read_csv("/Users/swastik/Downloads/restaurant.csv")
reviewdf=pd.read_csv("/Users/swastik/review.csv")
reviewerdf=pd.read_csv("/Users/swastik/Downloads/reviewer.csv")

# Mapping Reviews dataset to Restaurant Dataset
mergedDf = pd.merge(reviewdf,restaurantdf[["restaurantID","reviewCount","rating"]],
                    how="left",left_on="restaurantID",right_on="restaurantID")
mergedDf.columns

# Mapping Merged Reviews dataset to Reviewer Dataset
mergedDf =
mergedDf.rename({'rating_x':'reviewRating','rating_y':'restaurantRating'},axis='columns')
finalDf = mergedDf.merge(reviewerdf[["reviewerID","friendCount",
                                     "complimentCount","tipCount","fanCount"]],
                    how = "left",left_on = "reviewerID",right_on = "reviewerID")
finalDf.columns

finalDf.head()



#Handling Null Values
count=finalDf.isnull().sum().sort_values(ascending=False)
percentage=((finalDf.isnull().sum()/len(finalDf)*100)).sort_values(ascending=False)
missing_data=pd.concat([count,percentage],axis=1,
                       keys=['Count','Percentage'])
print('Count and percentage of missing values for the columns:')
missing_data

#dropping null values
finalDf.dropna(how='any').shape
finalDf.info()




#Adding a new columns based as true and fake reviews.
finalDf['flagged'] = np.where(np.logical_or(finalDf['flagged']=='NR' ,
finalDf['flagged']=='N'), 0, 1)
finalDf['flagged']
```


```
def specific_class(c):
    if c['flagged'] == 0:
        return "TRUE"
    else:
        return "FAKE"

finalDf['given_class'] = finalDf.apply(specific_class, axis=1)
print(finalDf['given_class'])

finalDf.flagged.value_counts()/len(finalDf)


#Randomize data to get equal percentage of real and fake reviews.
trueDf= finalDf[finalDf['given_class'] == 'TRUE']
trueDf= trueDf.sample(frac=0.10,random_state=8)

fakeDf= finalDf[finalDf['given_class'] == 'FAKE']
fakeDf= fakeDf.sample(frac=0.10,random_state=8)

modelDf=pd.concat([trueDf,fakeDf])
modelDf.info()
modelDf.given_class.value_counts()/len(modelDf)


#subset data for 100000 dataset.
n = 100000
modelDf=modelDf.iloc[    :n]


#Pie Chart for true and fake reviews
fig, ax = plt.subplots(figsize=(4, 4))
patches, texts, pcts = ax.pie(pd.Series(modelDf.given_class).value_counts(),
labels=modelDf.given_class.unique(),
                              autopct='%.1f%%', wedgeprops={'linewidth': 2.0, 'edgecolor': 'white'},
                              textprops={'size': 'large'}, shadow=True, colors=['blue', 'salmon'])
plt.setp(pcts, color='white', fontweight='bold')
ax.set_title('Distribution of True and Fake Reviews', fontsize=13)
plt.tight_layout()

sns.set_palette("pastel")
barGraph = sns.countplot(data = modelDf, x = 'given_class', hue = 'reviewRating')
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()


# initializing Stopwords
from nltk.corpus import stopwords
```


```
stop_words = stopwords.words('english')
stop_words.extend(['come','order','try','go','get','make','plate','dish','place','would','really','like','
great','came','got'])

data = modelDf.reviewContent.values.tolist()

#Tokenize words and Clean-up text
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  # deacc=True
removes punctuations

data_words = list(sent_to_words(data))


print(data_words[:1])

# Building the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher
threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

# See trigram examples
print(trigram_mod[bigram_mod[data_words[0]]])

# Define functions for stopwords, bigrams, trigrams and data lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc
in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out

# Remove the Stop Words
data_words_nostops = remove_stopwords(data_words)
```


```
# Form the bigrams
data_words_bigrams = make_bigrams(data_words_nostops)


# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])


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

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ',
'VERB', 'ADV'])

print(data_lemmatized[:1])

# Create Dictionary as id2word
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus(matrix)
texts = data_lemmatized

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

# View
print(corpus[:1])

# Building LDA model using ldamulticore

lda_model = gensim.models.ldamulticore.LdaMulticore(corpus=corpus,
                                    id2word=id2word,
                                    num_topics=10,
                                    chunksize=2000,
                                    passes=80,
                                    iterations=80,
                                    random_state=100,
                                    per_word_topics=True)

# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())

doc_lda = lda_model[corpus]

# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
vis
pyLDAvis.save_html(vis,'/Users/swastik/Downloads/lda13' + '.html')
```


```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model
is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized,
dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)



#compute coherence score for number of topics
def compute_coherence_values(dictionary, corpus, texts, limit, start=4, step=5):

    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.ldamulticore.LdaMulticore( corpus=corpus,
num_topics=num_topics, id2word=id2word)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary,
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=data_lemmatized, start=4, limit=30, step=5)

# Show graph
limit=30; start=4; step=5;
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()

# Print the coherence scores

for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))

# Visualize the topics

optimal_model = model_list[1]
model_topics = optimal_model.show_topics(formatted=False)
pyLDAvis.enable_notebook()
```


```
vis = pyLDAvis.gensim_models.prepare(optimal_model , corpus, id2word)
vis
pyLDAvis.save_html(vis,'/Users/swastik/Downloads/lda11.6' + '.html')

# Select the model and print the topics
#optimal_model = model_list[3]
#model_topics = optimal_model.show_topics(formatted=False)
#pprint(optimal_model.print_topics(num_words=10))

#Probabilities for each review
#maxIndex = mcvDf['Coherence Score'].idxmax()
#optCount = mcvDf['Number of Topics'].iloc[maxIndex]

datavec= []
for i in range(len(data_lemmatized)):
    finaltopics = optimal_model.get_document_topics(corpus[i], minimum_probability=0.0)
    topic_vec= [model_topics[i][1] for i in range(9)]
    datavec.append(topic_vec)

probVec = pd.DataFrame(datavec)

probVec['totalProb'] = probVec.apply(np.sum, axis=1)
probVec.head()

# Collate Probabilities with finalDf
modelDf = modelDf.reset_index(drop=True)
colNames = probVec.columns
modelDf[colNames] = probVec

# Load Dataset
classifyDf = modelDf
classifyDf.info()


 # Create Dataframe with Probabilities
train_array = np.asarray(probVecs)
print(train_array[3])

label_array = np.asarray(train_label)
print(label_array)

X_train, X_test, y_train, y_test =  train_test_split(train_array, label_array, test_size=0.20,
random_state=13)

#Logistic Regression model
clf = LogisticRegression(random_state = 10, penalty='l2',
                         fit_intercept=True,intercept_scaling=1,
                         class_weight=None,solver='lbfgs', max_iter=200,multi_class='auto',
                         verbose=0)
```


```
# y_train = y_train.astype('int')

clf.fit(X_train, y_train)
prediction_s3 = clf.predict(X_test)

#Decision Tree Classifier algorithm
clf = DecisionTreeClassifier(criterion='gini',splitter='best', max_depth=70,
                        min_samples_split=2,min_samples_leaf=50,
                        min_weight_fraction_leaf=0.0,
                        max_features='auto',max_leaf_nodes=None,
                        random_state=None,class_weight=None)
# Train the model on training data
# y_train = y_train.astype('int')

clf.fit(X_train, y_train)

# Use the forest's predict method on the test data
prediction_s2 = clf.predict(X_test)


#Random Forest classifier Algorithm
clf = RandomForestClassifier(n_estimators=500, criterion='gini', max_depth=70,
                        min_samples_split=2,min_samples_leaf=50,
                        min_weight_fraction_leaf=0.0, max_features='auto',
                        max_leaf_nodes=None,bootstrap=True, oob_score=False, n_jobs=1,
                        random_state=None, verbose=0, warm_start=False,

                        class_weight=None)
# Train the model on training data

clf.fit(X_train, y_train)

# Use the forest's predict method on the test data
prediction_s1 = clf.predict(X_test)

#logistic regression metrics
for i in range(len(prediction_s3)):
    if prediction_s3[i] < 0.400:
        prediction_s3[i] = 0
    elif prediction_s3[i] >= 0.400:
        prediction_s3[i] = 1

tn, fp, fn, tp = confusion_matrix(y_test, prediction_s3).ravel()
precision3 = tp / (tp + fp)
recall3 = tp / (tp + fn)
fmeasure3 = (2 * precision3 * recall3) / (precision3 + recall3)
accuracy3 = accuracy_score(y_test, prediction_s3)
print(precision3, recall3, fmeasure3, accuracy3)

print(classification_report(y_test, prediction_s3, target_names=['0','1']))
```


```
print(explained_variance_score(y_test, prediction_s3, multioutput='raw_values'))
fpr3, tpr3, thresholds0 = metrics.roc_curve(y_test, prediction_s3, pos_label=0)
aucscore3 = metrics.auc(fpr3, tpr3)
```

```
Output: precision    recall  f1-score   support

           0       0.60      0.89      0.72     11574
           1       0.55      0.19      0.28      8426

    accuracy                           0.69     20000
   macro avg       0.58      0.54      0.50     20000
weighted avg       0.58      0.59      0.53     20000
```

#decison tree metrics

```
for i in range(len(prediction_s2)):
   if prediction_s2[i] < 0.400:
      prediction_s2[i] = 0
   elif prediction_s2[i] >= 0.400:
      prediction_s2[i] = 1

tn, fp, fn, tp = confusion_matrix(y_test, prediction_s2).ravel()
precision2 = tp / (tp + fp)
recall2 = tp / (tp + fn)
fmeasure2 = (2 * precision2 * recall2) / (precision2 + recall2)
accuracy2 = accuracy_score(y_test, prediction_s2)

print(classification_report(y_test, prediction_s2, target_names=['0','1']))
print(explained_variance_score(y_test, prediction_s2, multioutput='raw_values'))
fpr2, tpr2, thresholds0 = metrics.roc_curve(y_test, prediction_s2, pos_label=0)
aucscore2 = metrics.auc(fpr2, tpr2)
```

```
oputput  precision    recall  f1-score   support

           0       0.61      0.71      0.66     11574
           1       0.48      0.37      0.42      8426

    accuracy                           0.67     20000
   macro avg       0.55      0.54      0.54     20000
weighted avg       0.56      0.57      0.56     20000
```

#random forest metrics

```
for i in range(len(prediction_s1)):
   if prediction_s1[i] < 0.4:
      prediction_s1[i] = 0
   elif prediction_s1[i] >= 0.4:
      prediction_s1[i] = 1
```

```
print(prediction_s1)
tn, fp, fn, tp = confusion_matrix(y_test, prediction_s1).ravel()
precision1 = tp / (tp + fp)
recall1 = tp / (tp + fn)
fmeasure1 = (2 * precision1 * recall1) / (precision1 + recall1)
accuracy1 = accuracy_score(y_test, prediction_s1)
print(accuracy1)

print(classification_report(y_test, prediction_s1, target_names=['0','1']))
print(explained_variance_score(y_test, prediction_s1, multioutput='raw_values'))
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test, prediction_s1, pos_label=1)
metrics.auc(fpr1, tpr1)
fpr1, tpr1, thresholds0 = metrics.roc_curve(y_test, prediction_s1, pos_label=0)
aucscore1 = metrics.auc(fpr1, tpr1)
```

```
precision    recall  f1-score   support

           0       0.61      0.84      0.71     11574
           1       0.55      0.28      0.37      8426

    accuracy                           0.70     20000
   macro avg       0.58      0.56      0.54     20000
weighted avg       0.59      0.60      0.56     20000
```