DECODING FACTORS IN ALTRUISTIC REQUESTS LEADING TO SUCCESS USING MACHINE LEARNING AND NLP

A PROJECT REPORT

Submitted by

SIDDHANT THAKUR [Reg No: RA1811027010004]

Under the guidance of

Dr. M. Ramprasath

(Assistant Professor, Department of Data Science and Business Systems)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled "**DECODING FACTORS IN ALTRUISTIC REQUESTS LEADING TO SUCCESS USING MACHINE LEARNING AND NLP**" is the bonafide work of "**SIDDHANT THAKUR [Reg No: RA1811027010004]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

GUIDE
Dr.M.Ramprasath
Assistant Professor
Dept. of Data Science and Business Systems

PANEL HEAD
Name of the Panel Head
designation
Dept. of Computing Technologies

CO-GUIDE(if any)
Name of the Co-guide
designation
Dept. of ——

HEAD OF THE DEPARTMENT
Dr. M. Lakshmi
Professor
Dept. of Data Science and Business Systems

INTERNAL EXAMINER

EXTERNAL EXAMINER

SRM Institute of Science & Technology
School of Computing
Department of Computing Technologies

Own Work Declaration Form

This sheet must be filled in and signed with date along with the student registration number, work will not be marked unless this is done.

To be completed by the student:

Degree/ Course :

Student Name :

Registration Number :

Title of Work :

I / We hereby certify that this work compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this project is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

Signature of the Student(s)

ACKNOWLEDGEMENTS

I express my humble gratitude to **Dr.C. Muthamizhchelvan**, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. I extend my sincere thanks to Dean-CET, SRM Institute of Science and Technology, Dr. T.V.Gopal, for his invaluable support. I extend my sincere thanks to **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support. I wish to thank **Dr. M.Lakshmi**, Professor & Head, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

I want to convey my thanks to my Panel Head, **Dr. S.Ganesh Kumar**, Associate Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

I register my immeasurable thanks to my Faculty Advisor, **Dr. Priyadarsini K**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete my course.

My inexpressible respect and thanks to my guide, **Dr. M.Ramprasath**, Assistant Professor, Department of Data Science and Business Systems for providing me an opportunity to pursue my project under his mentorship. He provided me the freedom and support to explore the research topics of my interest.

I sincerely thank the staff and students of the Data Science and Business Department, SRM Institute of Science and Technology, for their help during my project work. Finally, I would like to thank my family members and my friends for their unconditional love, constant support and encouragement.

Siddhant Thakur

ABSTRACT

Altruism is the definition of having a selfless concern for someone else's well-being. Social media has now established a platform where people can be altruistic through different online philanthropic communities and question & answer sites like Kickstarter or StackOverflow. Thousands and thousands of requests and questions are recorded on these sites, and people donate to or answer these pleas without considering their monetary benefit. An example of one of these online communities is the r/Random_Acts_Of_Pizza subreddit on the popular social discussion forum Reddit. This subreddit is an online community where any user can post a request for a free pizza while stating the condition they are currently in, and the outcome of each request is either the user was successful or unsuccessful in getting a pizza. This paper attempts to understand the factors that lead to a request being successful or unsuccessful by proposing a new architecture of stacking two models, one dealing with the sparse vector generated from text and the other dealing with the dense features gathered by looking at previous works, in order to predict the success of a request. After implementing the architecture, we find that the probability estimated from the first model and the number of comments on the request post plays an essential role in predicting the outcome of a request.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 History of Natural Language Processing

Natural Language Processing has been one of the key sub application of the Machine Learning and Deep Learning ecosystem and is used particularly to deal with the textual section of data. It has widespread use cases under it just like problem statements dealing with images.

1.1.1 General History

Natural language processing emerged as a discipline after World War II in the 1940s. Individuals recognised the importance of language translation at the time and attempted to construct a computer capable of doing this function automatically. However, it is evident that the task was not as straightforward as first believed. The volume of information transmitted through text has been staggering. Textual data has become one of the most essential, if not the most important, sources of information in recent years. Attempts to extract information from text (or voice) date all the way back to the 1950s, when scientists successfully translated 60 Russian lines into English and made some headway toward machine translation in the Georgetown-IBM Experiment.

1.1.2 From the 1950s

By 1958, a number of scholars had recognised major barriers to the development of natural language processing. Noam Chomsky was one of these academics who found it odd that nonsense statements that were grammatically correct were deemed as

insignificant as nonsense sentences that were not grammatically right. Chomsky objected to the sentence "Colorless green ideas sleep furiously" being classified as equally improbable as "Furiously sleep green colourless ideas"; any English speaker would recognise the former as grammatically correct and the latter as incorrect, and Chomsky believed that the same should be expected of machine models.

John McCarthy introduced the programming language LISP (Locator/Identifier Separation Protocol) in 1958, a computer language that is still in use today. ELIZA, a "typewritten" remark and answer procedure established in 1964, was supposed to mimic a psychiatrist's use of reflection methods. (It accomplished this by rearranging phrases and adhering to very basic grammatical rules, but the machine lacked comprehension.) Additionally, the National Research Council (NRC) of the United States established the Automatic Language Processing Advisory Committee, or ALPAC, in 1964. This group was charged with assessing the state of research in NLP.

In the year 1966, the National Research Council and Automatic Language Processing Advisory instituted the first artificial intelligence and natural language processing freeze, by ceasing funding for research in natural language processing and machine translation. Machine Translation has remained as one of the more costly application than just basic human translation after 12 years. Besides that, no computer was capable of carrying on a simple discussion in this application. In 1966, many believed that Artificial Intelligence and Natural Language Processing (NLP) research had reached a stalemate (though not all).

1.1.3 From the 1960s

Between 1957 and 1970, researchers in the field of natural language processing divided into two camps: symbolic and stochastic. Symbolic, or rule-based, researchers concentrated on formal languages and syntax generation; this group included a large number of linguists and computer scientists who saw this line of study as the origin of

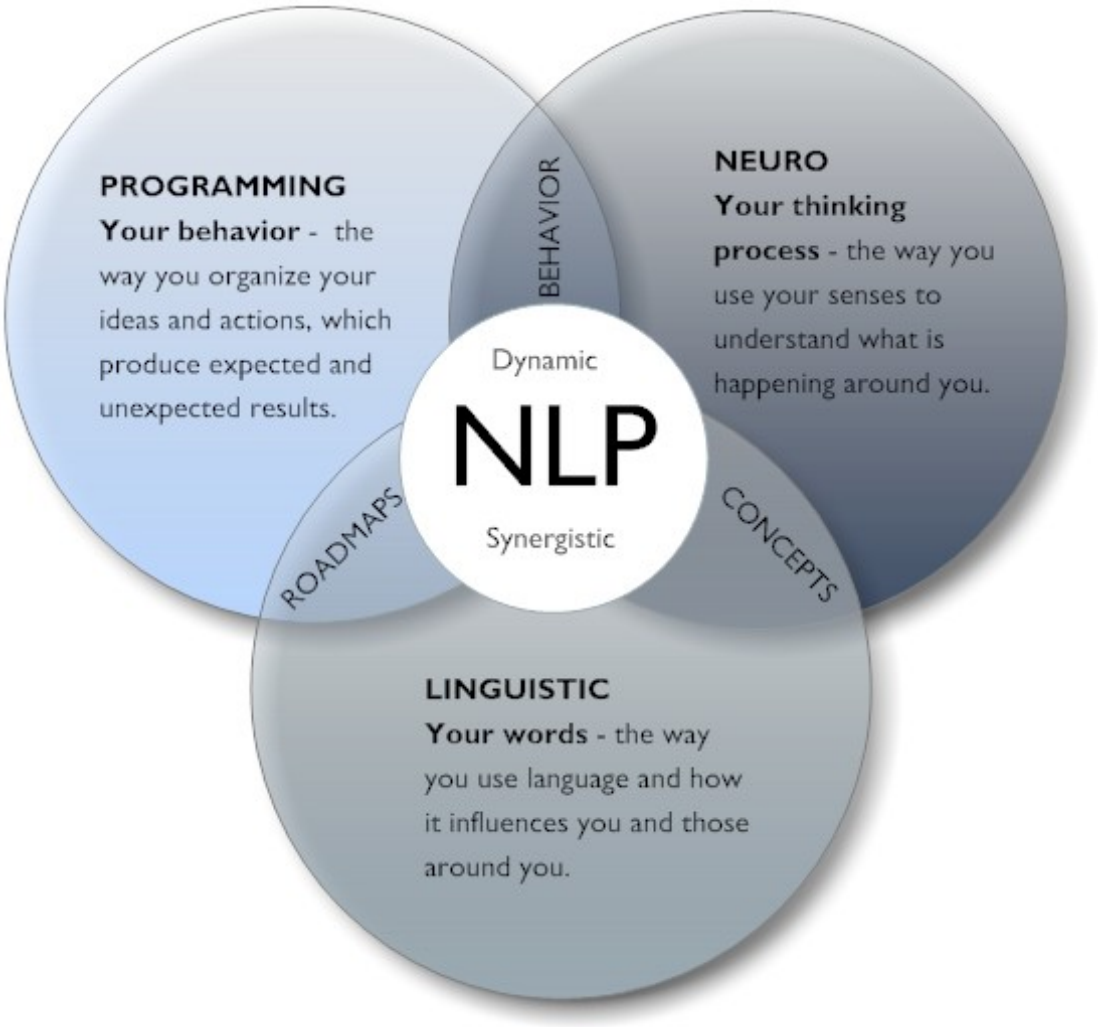

Figure 1.1: Composition of Natural Language Processing

artificial intelligence. Stochastic researchers became increasingly interested in statistical and probabilistic approaches to natural language processing, particularly in the areas of optical character recognition and pattern detection between texts.

1.1.4 From the 2000s

The amount of textual data shared ever since the dawn of the internet has been appalling. In recent times, textual data has been one of the most important, if not the most important, sources of information. Until the 1980s, the research work under Natural Language Processing was based on sets of hand-written rules, which became obsolete after the introduction of machine learning algorithms to textual data. The beginning of statistical NLP gave way to the advancement of many different subfields like machine translation, text classification, speech recognition, etc. Further, the need for up to the minute results in many of these tasks gave rise to what is today known as Neural Natural Language Processing or Neural NLP [2]. While utilizing deep learning methods, Neural NLP has become increasingly essential for sectors like healthcare and medicine.

1.2 Use-Case in Computational Social Science

Computational social science is primarily concerned with the data driven examination for different social networks that are complicated in nature, which usually necessitates expertise in analysis graph networks, natural language processing and machine learning. As a result, this lesson is organised around 4 important subjects that lie under the umbrella of the developing area of computational social science: Psychological & sociological sciences; natural language processing; complex networks; big data and machine learning.

Use cases in Computational Social Science each needs highly advanced natural

language processing technologies to handle. If it's judging whether news is skewed, recognising the emotion underlying tweets, or analysing the persuasiveness of an argument in a debate, the usage of NLP tools for social causes has certainly risen. The advent of social media and the application of Natural Language Processing in social settings has resulted in the emergence of philanthropic groups on the internet.

The fast growth of these websites has successfully created a stronger basis for comprehending the public's dislikes and preferences. These sites are often used to exchange ideas and opinions on a variety of issues, both uncritical and socially critical. Additionally, these services might be used to ascertain the societal reaction to certain situations. Individuals may exchange stories about natural catastrophes, learn about the influence of freshly released films, and express their thoughts about recently made items. These good developments in microblogs and variations in public evaluation spawned the field of opinion mining, which has been demonstrated to be critical for comprehending the people's attitude and current trends.

Recently, a wave of use-cases in Computational Social Science has surfaced, requiring state-of-the-art NLP tools to solve them [3]. Be it determining biased news, identifying emotion behind tweets, or assessing the persuasiveness of an argument in a debate, the use of NLP tools for social causes has clearly increased.

With the rise of social media and the use of Natural Language Processing in a social setting, the rise of philanthropic communities on the internet is notable. Websites like Kickstarter, Ketto, Reddit, and Kiva have made their presence immense and facilitated many people worldwide to get out of unpleasant situations via crowdfunding. For example, the Indian online crowdfunding platform, Ketto, has been responsible for raising over Rs. 2000 crores across 3.2 Lakh online fundraisers. This new era of online fundraising has undoubtedly helped people around the world and will certainly continue to do so. My project focuses on a similar community forum, under the Reddit website, known as r/Random_Acts_Of_Pizza [4]. The forum was created in the year 2010 with the aim of getting pizza to those who need it. Requesters

could explain their situation in a post, giving information about why they need a pizza and how it would help them. It would then depend on the other users to donate a pizza to the requester.

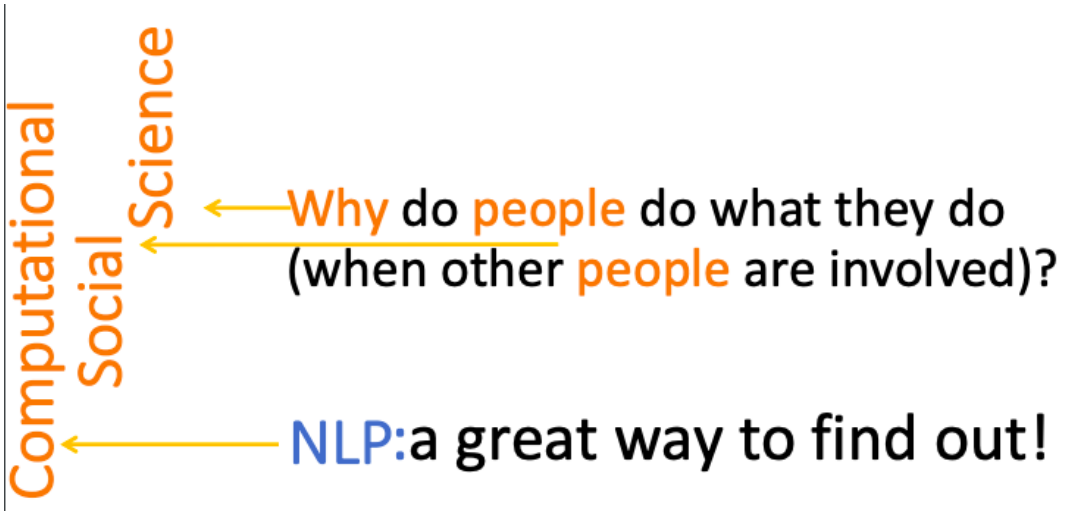

Figure 1.2: Computational Social Science

1.3 Random_Acts_Of_Pizza - A Subreddit

Figure 1.3: RandomActsOfPizza

1.3.1 History of Random Acts of Pizza Subreddit

There are many different subreddits on the website Reddit that have helped a lot of different people just by being a part of an altruistic community. Random Acts of Pizza is another one of these altruistic communities that helped a lot of people in feeding themselves or their loved ones.

In the year 2010, Redditor Gadianton founded the subreddit /r/Random Acts Of Pizza, which grew to nearly 24,600 followers over the next three years. On June 18th, 2011, the site RandomActsOfPizza.com was established, complete with an explanation of the subreddit's operation, instructions for sending and receiving pizza, and a verification mechanism. A Facebook page named "Random Acts of Pizza" was started on July 2nd. On Thursday, October 4th, @random pizza A Twitter account was formed, which was used to republish many of the subreddit's posts. On October 12th, 2012, Gadianton, the originator of RAOP, uploaded a snapshot of a document (seen below) confirming that Random Acts of Pizza, Inc. had been founded as a first step toward establishing a tax-exempt charity.

1.3.2 Dataset

The dataset used for this project was taken from the CS Stanford website, and the foundation for this project has been the work done by Althoff et al. [5]. The dataset contains 5671 requests collected from the Reddit community between December 8, 2010, and September 29, 2013. There are 33 features present in the dataset, including a target variable containing the information about whether the requester received the pizza or not. The project proposes an architecture of a two-stack binary classification model to predict the success of an altruistic request. The first model deals with the textual data by utilizing textual feature extraction techniques to convert the raw text data into vector form and then classify the vector data against the target variable. The finished model is then used to estimate the probabilities of the request being successful by the text. This probability estimate is then used as a dense feature among other

features, like account_age of the requester, the number of upvotes and downvotes, etc., to predict the outcome of the request. We have then used the final model to determine the importance of each feature and how they contribute towards predicting the target variable.

CHAPTER 2

LITERATURE REVIEW

The work covers a broad spectrum of classic NLP tasks, with more specialised systems being underpinned by general-purpose syntactic and semantic algorithms. We are specifically concerned with the algorithms that have a good scalability and hence can be effectively implemented in a dispersed setting.

Syntactic systems anticipate the parts-of-speech for each of the word present in the sentence. Additionally, these systems identify connections and relations between words, such as subject of the sentence, object of the sentence, modification present in the sentence for the verb, and so on. The emphasis is on efficient algorithms that make use of vast volumes of unlabelled data and, more recently, on neural network automation.

On the semantic side, we recognise the things present in free text, classify them according to their category (such as person, place, or organisation), cluster their mentions inside and over texts (which may also be know as coreference resolution), and work on resolving the entities related to the Graph of Knowledge.

Recently, the work done in natural language processing has focused on incorporating many sources of data and information to facilitate text analysis and the use of semantics at different levels like sentence, document, etc.

In recent years, along with the increase in the research work under NLP in a social setting, there have been multiple attempts to decode linguistic factors in textual data and how they affect the success of requests.

2.1 Base Paper

With the objective of interpreting what makes a favor successful, Althoff et al. [5] discuss high-level social factors, like the status of the requester and the similarity between the requester and the donor, textual factors, like evidence of need and the sentiment of text, and, finally, temporal factors like age of the account and the time the request was made. A logistic regression model then binds these factors together to classify whether the request was successful or not. The model also allows the researchers to reason out the significance of independent variables in predicting the dependent variable.

The project develops an architecture for accounting for each of these possible confounds while investigating the significance of social variables and language variables.

The paper also demonstrates using a prediction task that the proposed model generalized to requests that it has not seen before and significantly improves over several baselines. Through the model, it finds out the importance of each feature, where including image in sort of evidence and having posted in the subreddit before increases the chances of a request being successful.

By including all the factors like Temporal, Social, Text and Unigram, the project received it's best AUC score of .672, which sets a baseline for all the other future projects.

2.2 Addition of Sentiment-Based Feature

Similar to the base paper, Jacek and Sabrina et al. [6] focuses on the comparative analysis of different machine learning algorithms over the Random Acts of Pizza

dataset. In addition to the social and linguistic factors, such as the requester's identity and how the requester is asking, the researchers also aim to extend the work by investigating the sentimental factors satisfying a donor.

This project goes against the belief held by the base paper that positive sentiment and politeness are associated with success. By using the polarity calculation present in the NLTK API, it gets the numeric value for polarity enclosed in the text.

The project's goal was to find out the emotions present in the textual requests in order to find what emotions distinguish a successful request. This initiative contradicts the foundation paper's assertion that happy feelings and politeness are linked to success. It retrieves the numeric value for polarity contained in the text by utilising the NLTK API's polarity computation. The text analysis was done by using the O2MC Api that allowed the extraction of emotions like Joy, Trust, Sadness, Disgust, etc. Based on the analysis most of the successful text requests had Joy as the dominant emotion, with Trust and Fear behind it.

2.3 Use of a Graph-Based Predictor

H.-P. Hsieh et al. [7] extend the base paper by introducing additional features like Topic, Role, and Centrality. These features are useful in handling the hidden meaning behind the text by using Bag of Words & N-Gram and the role of interaction of the person requesting the pizza with other users present in the community. It then uses a model named the Graph-based Predictor for Request Success model (GPRS).

It uses a Graph and a Propagation based Optimization algorithm to identify the similarity-based correlation between requests present in the data and learn the weights to compute probabilities of successful or unsuccessful requests, such that the requests having higher similarity scores will have the same label. Ahmed et al. [8] discuss the

same method, focusing more on the Topic and Role as additional features.

In Centrality, the paper proposes a directed weighted interaction graph which represents their interaction behaviors where each node is a user and each directed edge refers to a comment from the user to the recipient.

The features categorized under the Role category measure the use of interaction of the person requesting the pizza among the other users in terms of the densely or loosely connected nodes in a graph. These are based on different social theories.

Among Topic features, the paper uses three kinds of ways to understand the hidden topics, BOWs or Bag Of Words, N-gram and LDA (Latent Dirichlet Allocation) Hidden Topic. They aim to extract the hidden topic present in the requester's text.

2.4 Extended Dataset to Understand Persuasive Requests

Durmus and Cardie et al. 's [9] work is based on debate.org, which provides a platform for people to express their opinions and beliefs. It takes on the task of judging which debater will be more successful. The paper provides a new dataset to study the effect of language use versus the prior beliefs on persuasion while proposing a controlled setting taking the political and religious ideology of the reader.

It inculcates user-based features like ideology and opinion similarity and linguistic features like text length, sentiment, and features gathered using TF-IDF.

Although based on a different dataset, i.e., the public debate dataset, the paper focuses on how persuasion can change a user's prior belief. It also suggests to explore

the effect of different aspects of people's background on persuasion.

2.5 Using Deep Learning to Tackle Persuasive Requests

Yang et al. [10, 11] also focus on modeling persuasive strategies using a semi-supervised hierarchical neural network incorporating both word-level and sentence-level attention to quantify persuasiveness.

The paper focuses on easing the quantification of persuasiveness of requests and generalizing persuasive strategies learned from one domain to another. The project leverages the partially-labeled data that is available to predict the persuasive strategies related for each of the sentences, where the learning comes from both the overall document-level labels and very limited sentence-level labels.

It uses data from different sites like Kiva, a lending platform, Random Acts of Pizza, an altruistic platform, and Borrow, a money-lending platform charging interest. Using this combination of data, it created a hierarchical weakly-supervised latent variable model to predict the sentence-level strategies for persuasion, supervised by the global or document-level labels.

2.6 Text Classification Survey

This survey addresses the growth in text classification problems in the recent years and the many applications developed to address them. They widely divided the process followed by text classification and document categorization into 4 phases, Feature Extraction, Dimension Reductions, Classifier Selection, Evaluation.

2.6.1 Feature Extraction

Due to the unstructured nature of texts and documents, there is a need to clean the data by omitting the unnecessary characters and words and apply formal feature extraction methods. The paper discusses the Term Frequency & Inverse Document Frequency (TF-IDF), Word2Vec, and GloVe. The survey discusses the technical implementation of these embedding models.

2.6.2 Dimensionality Reduction

As text or document data contains multiple unique words, the amount of time spent on pre-processing the data increases leading to a slower pipeline functioning. The survey suggests solutions like using inexpensive algorithms but also states that the use of cheap algorithms result in a bad performance on some datasets.

Therefore, the survey outlines a few common methods for reducing dimensions in the data like Principal Component Analysis or PCA, Linear Discriminant Analysis or LDA and non-negative matrix factorization or NMF. These methods were not used for this project but can be considered for future improvement of the architecture developed in this project.

2.6.3 Classifier Selection

The survey also classifies this step as one of the most important step of the text classification routine and also mentions that in order to choose the best algorithm, one needs to have a complete understanding of the algorithm theoretically.

With the advancement in natural language processing, the paper states that Logistic Regression, Naive Bayes Classifier, Support Vector Machines and Tree-based

classifiers are few of the essential machine learning algorithms present to understand features gathered from textual data and develop models. It also covers the deep learning algorithms like Recurrent Neural Networks or RNN, Long-Short Term Memory or LSTM and even Convolutional Neural Networks or CNN, that have achieved great results in comparison with the preceeding machine algorithms and continue to do so for a lot of the similar tasks that we have faced in the problem statement.

2.6.4 Evaluation

The survey suggests that even though accuracy calculation is one of the simplest evaluation methods available, it fails to work for unbalanced datasets. Therefore, the other alternatives are outlined, which include F-Score, AUC score, etc.

The paper also discusses the drawbacks and limitations of each of the methods listed above which helped me to gather the information required to work on this research project.

CHAPTER 3

PROPOSED ARCHITECTURE AND STEPS

This project utilizes the crucial features gathered from the pre-existing dataset as well as the previous works in the literature review while framing my architecture consisting of a stack of two models to deal with the textual and numerical data separately. In this section, I will be stating the architecture developed and process followed in creating the two models.

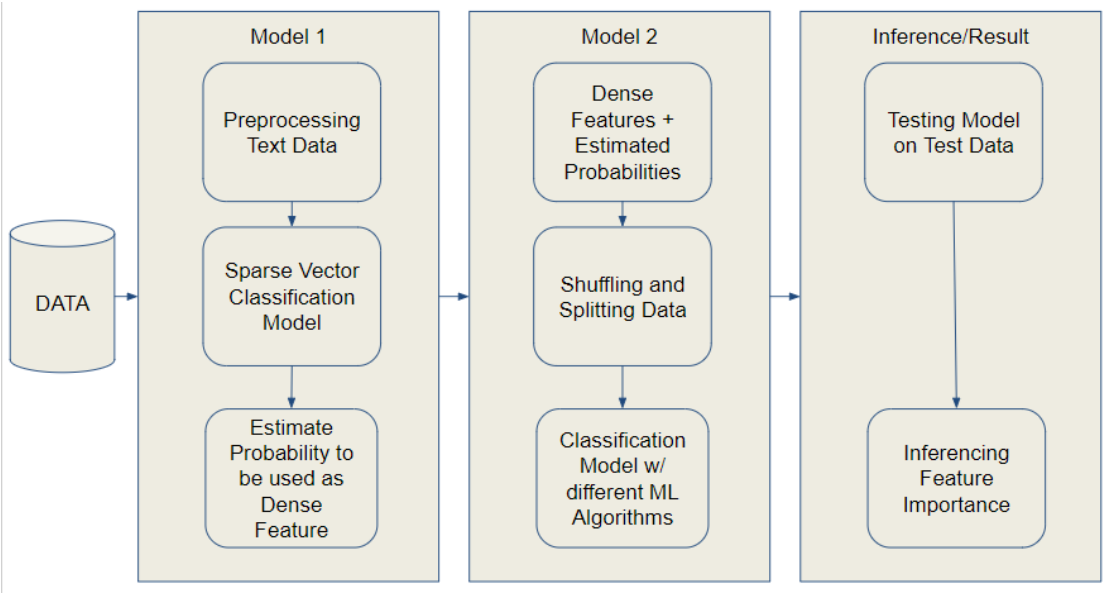

Figure 3.1: Model Architecture

3.1 Workflow

The project's workflow starts with the dataset itself by getting it from the Stanford's Website and preprocessing it for the model and the developer to better understand how to deal with the data. We then divide the dataset on the basis of whether the feature is textual or numeric as we need to work separately on them for my use-case.

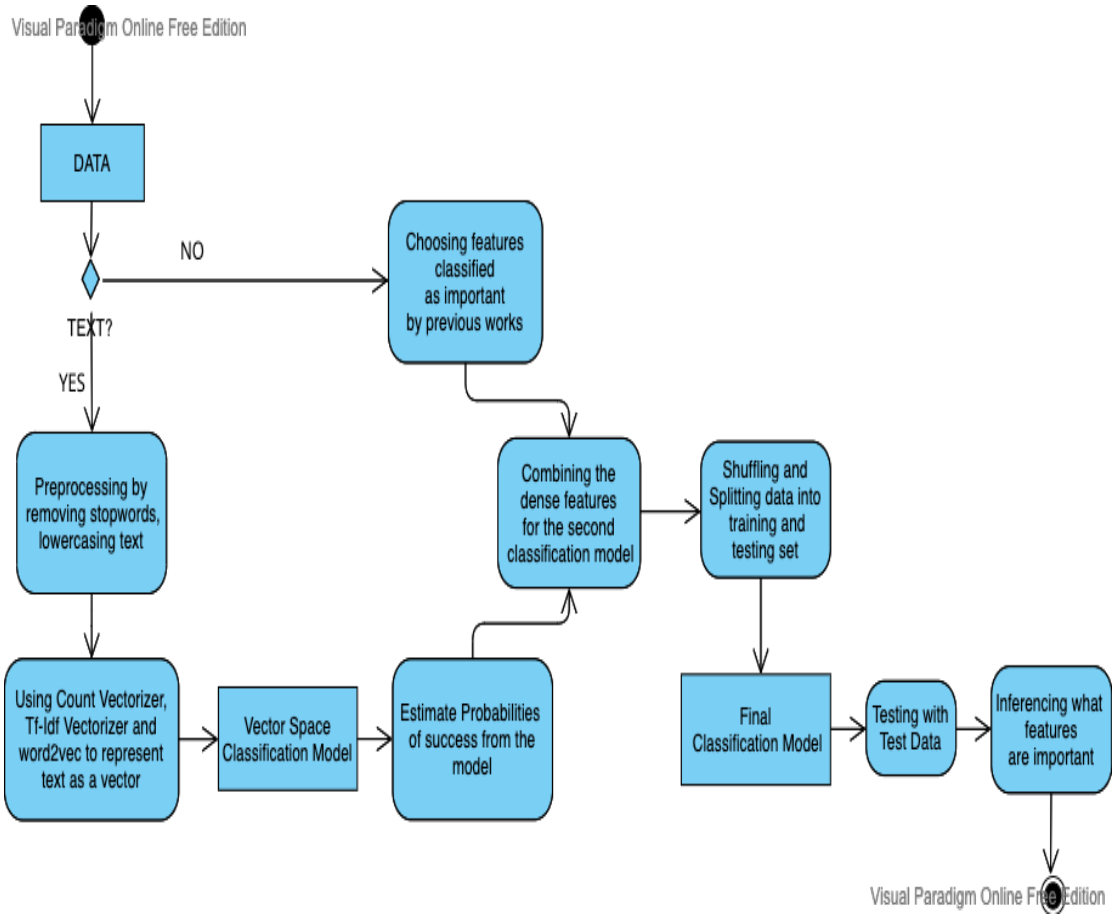

Figure 3.2: Workflow of the Model

If the feature is textual, we then proceed to preprocess it by following the basic preprocessing steps in text classification taking care if lowercasing everything might change the meaning, where casing is some time very important for Machine Translation. Therefore, we take care of all the text preprocessing with these features. For the other features which are numeric, we select features deemed important by previous research work done on this subject and dataset.

The preprocessed textual data is then fed to CountVectorizer, TF-IDF Vectorizer, a Min-Max Word2Vec model, and a Doc2Vec model in order to get the features from the text data and represent it as a vector for the machine learning model to make some sense out of it. We then create different models over the different vector embeddings and get a probability estimate to be used with the numeric data gathered previously.

We then combine the features for the second classification model to assess and act on it. The second classification model is the final model, it allows us to reduce the high dimensionality vectors of the text to a probability of being successful in requesting a pizza and use it with other dense features to ultimately predict whether the request is successful or not.

We then make use of the the Shapley values using the SHAP python package to get an understanding of how important and what their contribution is towards predicting the success of a request in the final mode.

3.2 Data Exploration

Data exploration allows for a better understanding of the dataset, making it easier for us to navigate the data better and select methods suitable for that data. Data exploration is the first stage of data analysis, during which a data developer can employ statistical tools and techniques to define dataset characteristics such as amount,

size, and correctness in order to get a better understanding of the data's behaviour. This helps in defining a set of boundaries under which we can work and get results for the problem at hand. They can also make use of different visualization softwares present on the internet to understand the data.

We can make use of both the manual analysis techniques and automated data exploration softwares that visually discover and identify the links between various data features, the organization of the dataset, the presence of values that diverse from the normal or are an anomalous, and the distribution of the values in the columns in order to identify the trends present in the dataset and base my hypothesis based on that, thereby enabling the user dealing with the data to gain a deeper and better understanding of the raw data.

After a simple exploration of the Random Acts of Pizza dataset, we notice that there are 5671 requests with an average success rate of 24.63%, making it an imbalanced dataset. This will allow us to choose Stratified K-Fold instead of the standard K-Fold Cross Validation technique while training my models. Stratified K-Fold Cross Validation partitions the dataset so that the validation set will have an equal number of instances of the dependent variable [12]. This ensures that no instance will overpower the other in the validation set.

3.3 Text Preprocessing

Preprocessing text simply means to transform it into a predictable and analyzable format for my work. There are multiple ways to preprocess the data, some of which have been followed to create the model architecture.

3.3.1 Lowercasing, Removal of Punctuation & Stop-words

Lowercasing, Removal of Punctuation & Stop-words are few of the most common preprocessing steps that allow us to reduce the redundancy between words. Similarly, Punctuation in a text can sometimes create difficulty in comprehending the text or extracting tokens. Therefore, it becomes necessary in some use cases to remove the punctuation in text.

Stop-words consist of the most common words used in any language and are usually removed from the text as they do not add any unique value to the text analysis. Therefore, we use the NLTK [13] list of English stop-words in my project to remove them from the request.

3.3.2 Regular Expression

Regular expressions, or RegEx, are a collection of characters that are mostly used to locate and replace patterns in text. Simply put, a regular expression is a collection of characters or a pattern that is used to locate substrings inside a given text.

A regular expression (RE) is a programming language that is used to describe text search strings. It enables us to match or extract additional strings or collections of strings by using a pattern's specific syntax.

Regular Expression [14] has proven powerful in searching and manipulating text strings and has been an essential tool in text analytics. For example, in the project, we used the "re" python library to find and expand the contractions such as "don't," & "can't," to "do not," & "can not."

3.3.3 Tokenization

Tokenization is the fundamental task of the NLP preprocessing pipeline. Tokenization is a technique for dividing a body of text into shorter components which can be called as tokens. These tokens might be words, letters, or subwords. Thus, tokenization can be generally categorised by three types: word tokenization, character tokenization, and subword tokenization.

In this project, we have used the "regexp_tokenize" function from the NLTK [13] library to tokenize the cleaned text.

3.3.4 Lemmatization

Lemmatization is the most crucial text preprocessing step that stems the word while ensuring it does not lose its meaning. The project uses of the Wordnet Lemmatizer present inside the NLTK [13] package. Wordnet [15] is a large lexical database that aims to establish structured semantic relationships between words. We utilize the NLTK [13] interface of this database using the function lemmatize() of the instance WordNetLemmatizer().

3.4 Feature Engineernig

Feature Engineering is the again one of the crucial steps in general modelling. Irrespective of the problem statement, features are required for every machine learning model to learn from. Features of a model are the facts that the machine learns and acts on. Features in a model help us predict the target variable, which can also come under a feature.

Sometimes we are usually fortunate enough to have pre-existing features present in

the dataset, however based on my domain knowledge and the research work we do in order to solve a particular problem statement can decide whether those pre-existing features are deemed useful or not. If they are not useful we then need to figure out ways to engineer more useful ones.

For example, in this project 3 or 4 of the features present in the dataset were actually useful and did not need any change. However, after completing the literature survey there was a definite need of adding more variables using the pre-existing ones. Therefore, for the numeric data we feature engineered the features like whether the requester had posted before or not or to simple get the length of the request text.

We split the numeric and textual data and separately work on it for the two different models.

3.4.1 Numeric Data

Based on the previous literature and knowledge, we manipulate the dataset in order to get the features. We use two features to understand how any posted request is doing in the community by subtracting the number of downvotes from the number of upvotes at retrieval as well as getting the number of comments at retrieval. We also add the length of the text as a feature to check if longer texts lead to the request being successful or not.

An "Evidence" feature is calculated by setting a value of 1 if there is a presence of any image link and 0 if there is no link present [5]. For other features, we consider the account age of the requester and whether the requester has posted before in the Random Acts of Pizza subreddit. These features give us some information about the requester.

3.4.2 Text Data

We combine the title of the request and the main text of the request to get the final text which can then be converted into vectors using One-Hot Encoding [16], TF-IDF [17], Word2Vec [18], and Doc2Vec [19] techniques.

3.4.2.1 One-Hot Encoding

One-Hot Encoding or Count Vectorizing is the method of representing words by creating a vector having as many dimensions as the unique words in the vocabulary. Inside the vector, if the text data features that word, we put a "1" in that dimension; otherwise, we put "0". This gives us a huge sparse vector representation of all the text requests in the data. However, one of the disadvantages of the Count Vectorizing technique is its inability to represent the semantic and statistical relationship between the data.

3.4.2.2 TF-IDF

TF-IDF vectors feature statistical representation of text regarding the word's importance or relevance in a document. This representation is constructive in information retrieval systems as it allows the system to look for more relevant words in the search query.

TF-IDF [17] measure is the product of Term Frequency(TF) [20] and Inverse Document Frequency(IDF) [21], where Term Frequency is a measure of the frequency of a word in a document, while Inverse Document Frequency is a measure of the importance of a word. Term Frequency can be defined as the ratio of a word's occurrence in a document to the total number of words present in a document. The IDF of a word is the natural log of the ratio of the total amount of documents in the corpus to the amount of documents that contain that word. Therefore, it gives

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

Figure 3.3: TF-IDF Formula

weightage to each word based on its frequency in the corpus.

3.4.2.3 Word2Vec

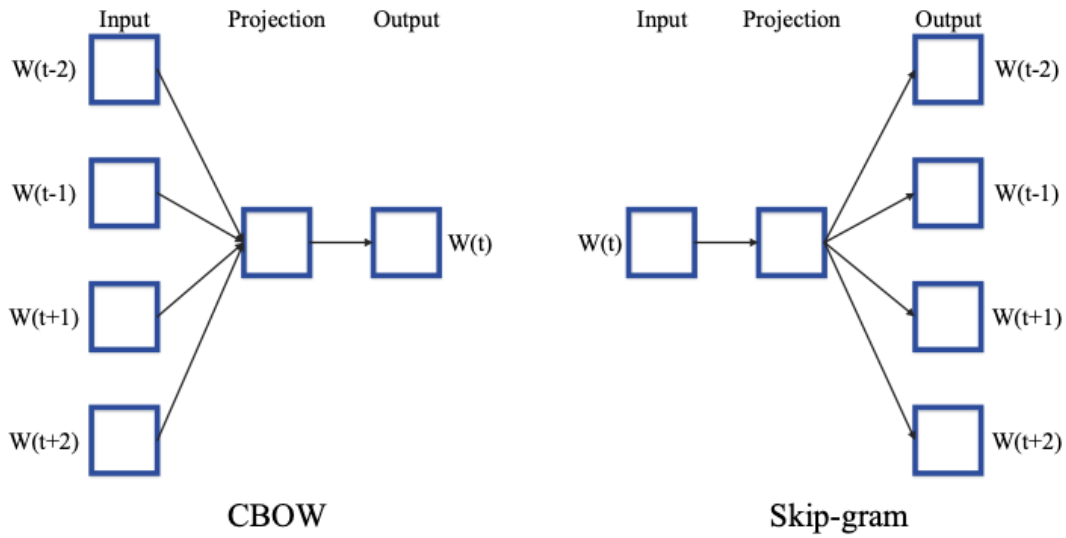

Figure 3.4: Word2Vec Architecture for both Continuous Bag-of-Words and Skip-gram

Word2Vec is another one of the frequently used embedding techniques which aim at learning embeddings for every word in the corpus such that the word vector in the

embedding space best captures the meaning of the word [22]. Word2Vec helps in capturing the semantic representation of the text. It uses neural networks to derive the word's meaning from its context. Therefore, Word2Vec projects the meaning of the word in a high-dimensional vector space and clusters the word with similar meanings together. We used the pre-trained Word2Vec model over the Google News Corpus for this project [18].

In order to find the sentence vector by using individual word vector representations, we calculate the average of the vectors of words in the text. Another method that has given good results in the past is calculating the minimum and maximum from the list of vectors of words in the text and concatenating it to form a more extensive vector representation [23].

3.4.2.4 Doc2Vec

Doc2Vec [19] is very similar to the Word2Vec [18] model in terms of its general architecture. In addition to learning the word vectors, it also learns the paragraph vector associated with the full text. There are two architectures of the Doc2Vec model called the Distributed Memory (DM) and the Distributed Bag of Words (DBOW). This project uses the Distributed Memory (DM) architecture of the Doc2Vec Model packaged under the Gensim library.

Rather than relying on just the surrounding words to anticipate the word, we introduce a document-unique feature vector to the architecture. Thus, as the word vectors W are trained, the document vector D is also learned, and at the conclusion of training, it contains a numeric representation of the document.

Word vectors and document Id vectors are used as inputs. The term "vector" refers to a one-dimensional vector of size 1xV. The dimension of the document id vector is 1xC, where C is the total number of documents. VxN is the dimension of the hidden

layer's weight matrix W. CxN is the dimension of the hidden layer's weight matrix D.

The model described in the above two paragraphs is referred to as the distributed Memory variant of the Paragraph Vector model (PV-DM).

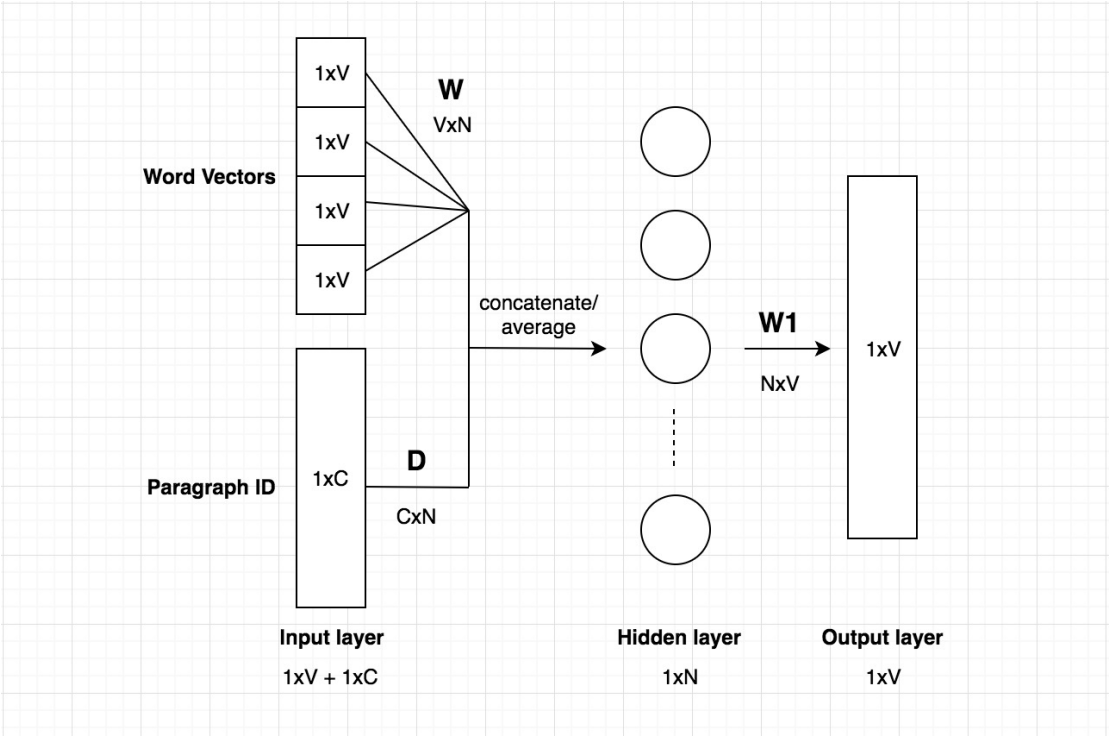

Figure 3.5: Doc2Vec Architecture

3.4.2.5 GloVe or Global Vectors for Words Representations

GloVe is another one of the effective word embedding approach that has been utilised for text categorization (GloVe). The strategy is quite the same as the strategy employed by the Word2Vec method, in which each word is represented by a high-dimensional vector and trained using the neighbouring words in a large corpus. Built on around 400000 vocabularies and learned using Wikipedia 2014 and Gigaword5 corpus, the word embedding is pre-trained and utilised in several works as well as the fifty dimensions for word presentation. Additionally, GloVe also offers a pre-trained word vectorizations with 100, 200, and 300 dimensions that have been trained on even larger corpora and fine tuned with some of the best models, that can include the Google news

as well as some of the tweets present from the website Twitter.

3.5 Modelling

This section briefly describes the stack of two models present in my classification architecture. The architecture consists of two models, one that deals with the high-dimensional sparse vector converted from the textual data and the other model that deals with the dense features pre-existing in the dataset as well as the ones engineered by previous literature.

3.5.1 Text Model

The request title and the request text are combined with the target variable into a separate data frame. This data frame will be helpful in converting each row of the text data into a vector using different text embedding techniques mentioned in section 3.3. The resulting vectors from the different embedding techniques are then fed into different machine learning algorithms like Logistic Regression [24], Gaussian Naive Bayes [25], and Random Forest [26]. (Note that we do not use K Nearest Neighbors [27] or Support Vector Machines [28] as they do not give a probability estimate needed for my second model.)

Based on the accuracy of the different models created by the combinations of different text embedding techniques and machine learning algorithms, we select the models to estimate the probability score for the success of the request and feed it to the final model.

3.5.2 Final Model

The final model consists of the numeric data present in the dataset, the derived features based on previous works, and the probability estimates of the text from the previously mentioned text models performing well. We use the Logistic Regression [24], Gaussian Naive Bayes [25], and Random Forest [26] algorithms to create a classification model using the target variable of receiving a pizza as the dependent variable and the other dense features the independent variable. In order to ensure that the model will do well on unseen data and prevent overfitting, we also implemented a Stratified K-Fold cross-validation method [12].

3.5.3 Machine Learning Algorithms

With the rise in the specifications of the machinery, Machine Learning algorithms can now be used to take advantage and understand the data that was previously heft to summarise. In the recent period, we have looked at the different innumerable advantages of machine learning methods, from using image to teach cars to auto drive to making recommendation engines that propose titles to watch based on watching histories and then to banks that detect fraudulent behaviour based on client spending patterns.

Machine learning has definitely been a helpful resource in enabling us to gather information and presenting insights by using the different algorithms developed to perceive the dataset. We are in an era where most of the automated machines around us are in line with the state-of-the-art machine learning methods that has made my lives much easier than it actually would have been. This also does not mean that it will replace humans as the machines are too dumb to unless and until we code them to be smart and that's where automation is peaking in it's research where people are now able to make the computer automatically code for dataset that it sees.

There are many serious advantages that machine learning and deep learning bring with them and we are heavily in debt to all of it even in my day to day lives. These algorithms are even capable of handling big and complicated datasets in order to extract interesting patterns or trends, such as outliers present in the dataset that go apart from the expected trend. That is what is usually used in capturing fraudulent behavior by seeing what is going out of the expected. Machines are, therefore, needed to evaluate the given data as fast as they can and make choices for the humans when it exceeds a certain level.

There are many machine learning algorithms which have been used for this project, listed below.

3.5.3.1 Logistic Regression

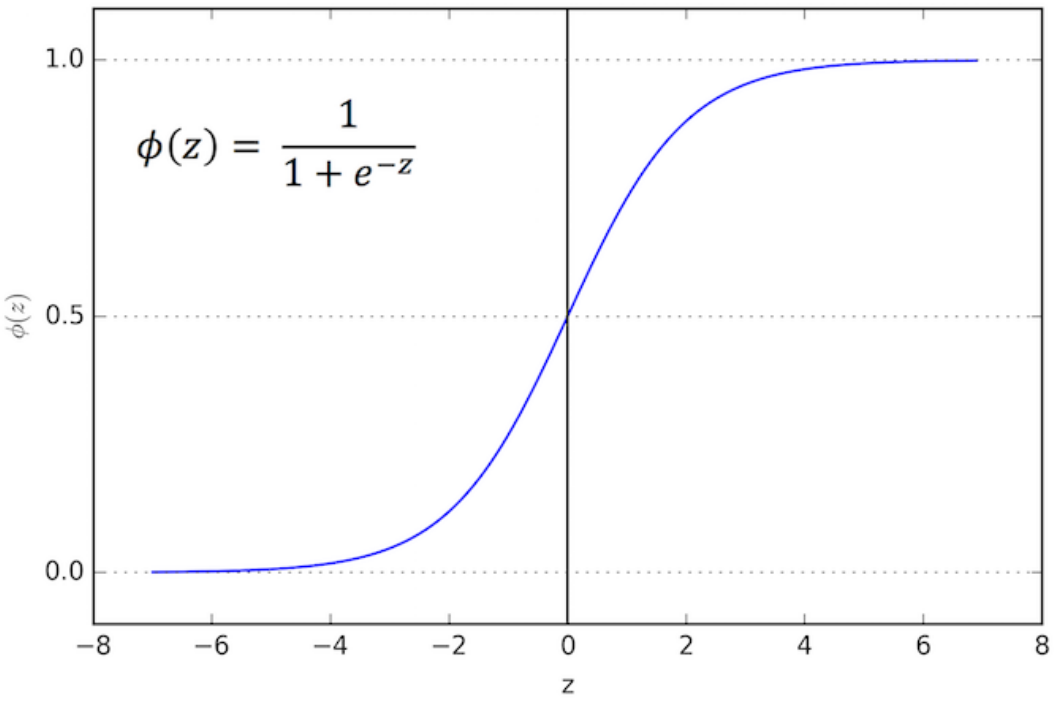

Figure 3.6: Logistic Function

Probabilities for categorization issues with two probable outcomes are modelled using logistic regression. It is a classification problem-specific extension of the linear regression model.

Logistic regression is a classification technique. Rather than fitting a straight line like linear regression, the logistic regression model squeezes the result of a linear equation between 0 and 1.

Logistic Regression is definitely one of the most simplest yet the most widely used algorithm when we're tackling classification. Even if we have a small amount of data available with us, we can make use of the logistic regression algorithm in order to at the very least set the baseline for my model which in some cases does not usually exceed without the addition of data.

The whole basis of the logistic function as shown in Figure 3.6 is the Sigmoid function (mentioned in the upper left corner of the graph shown in Figure 3.6) under which any value given to the function gives out a value between 0.0 and 1.0, thereby giving us a probability for my classification model.

3.5.3.2 Naive Bayes

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Figure 3.7: Bayesian Function

In the above formula, P(A) is the prior probability, P(B|A) is the likelihood, and P(A|B) is the posterior probability.

The Naive Bayes method is a kind of supervised learning that is based on the Bayes theorem and is used to solve classification issues. It is mostly used in text categorization, which requires a large training dataset.

The Nave Bayes Classifier is a simple and efficient Classification method that aids

in the development of rapid machine learning models capable of making accurate predictions. It is a probabilistic classifier, which implies that it makes predictions based on an object's likelihood.

It is termed Naive because it makes the assumption that the existence of one characteristic is unrelated to the occurrence of other traits.

3.5.3.2.1 Bayesian Statistics is the branch of statistics that believes in conditional probability and holds the ideology opposite of what is held by the frequentist school of statistics. In the formula mentioned above, the posterior probability is what we find out using the information present with us, i.e., the prior probability and the likelihood probability. The probability score which is used to divide the product of prior and likelihood is present to normalize the probability score for the posterior probability.

It is one of the most dependable statistical method in order to get a confidence interval or a distribution of probable values that can fit in a problem rather than depending on a point estimate.

3.5.3.3 Random Forest

Linear regression and logistic regression models do not work well when the relationship between the characteristics and the result is nonlinear or when the features interact. Tree-based models segment the data numerous times depending on specified feature cutoff values.

Random forest is one of the methods that is used for supervised classification machine learning. It develops a "forest" out of a multiple decision trees, which are trained using the "bagging" method. The bagging method's basic idea is to combine many learning models in order to improve the final output.

To state this in simpler words, what random forest does is that it inculcates multiple decision trees and then amalgamates them together to get a more robust & accurate statistical model.

More explanation on random forest is received from its origin, i.e., it is a Tree based method. We'll see more about decision trees now.

3.5.3.3.1 Decision Trees For classification and regression, Decision Trees or DTs are a parameter less supervised learning approach. The objective is to learn basic decision rules from data attributes to develop a model that predicts the value of a target variable.

A comparison value, a comparison operator, and an action are all condition branches in decision trees. Returning a result, continuing the examination, or stopping the evaluation are all options. A hierarchical tree structure organises the branches. At the tree's trunk, you usually identify common criteria and outcomes. The tree is then extended outward to include more specific situations and their effects. When the decision tree is used, the system starts with the top row and works its way down until it finds a result that is true. The system receives the outcome. If the system does not reach a returned result after processing all of the branches, it returns the final value otherwise.

3.6 Metrics

Metrics are necessary for us to understand the model and whether or not it is actually working correctly before even applying to use it for predictions. If the metrics based on the application turn out to be good then we can put the model to use.

Without the use of metrics we will not be able to test my model and let it deploy or use it as an application in the real world. We need to test my model on different

training and test set and then decide over if the model is ready to be deployed for full use.

Usually a good training metric and decent test metric is preferable for deploying. We can judge a good trade-off on the basis of the application or the problem being solved. Therefore, we can also judge whether the model is overfitting or underfitting.

3.6.1 Accuracy

It is the most often used performance statistic for classifier algorithms. It can be described as the proportion of accurate predictions to total predictions.
However, if there is a class imbalance present in the dataset, like in my case then accuracy becomes an unrealiable metric while measuring the performance of the model. Therefore, we use other metrics like Precision Recall, AUC Score.

3.6.2 Precision

Precision, which is critical for document retrieval, can be defined as the number of correctly retrieved documents by the machine learning model.

3.6.3 Recall

Recall is found out by dividing the number of true positive values by the total number of pertinent data, i.e., all the data that should have been identified as positive.

Precision and recall allow for a more accurate assessment of model performance in the presence of a class imbalance. However, if there is uneven class distribution, as precision and recall sometimes may start giving misleading results. Therefore, we use F-Score to maintain a tradeoff between precision and recall.

3.6.4 F-Score

In order to maintain a balance between precision and recall, so that they do not give misleading results, we use F-Score for my model's metric. F-score is the weighted average of the precision and recall, as it is the harmonic mean of the two.

F-Score has the highest value of 1 and on the other hand lowest value of 0.

3.6.5 AUC Score

Area Under Curve (AUC) is also a commonly utilised statistic for assessment. It is used to work on the problem statements involving binary categorization. The AUC score of a classifier is equal to the likelihood that when a randomly given positive sample will be ranked above than a randomly picked negative sample.

The results of the following model are given under Section 4 with Accuracy [29], Precision [29], Recall [29], and F1-Score [29] as the metrics.

3.7 Parameter Optimization

Parameter Optimization or tuning is the name given to the process of selecting parameter values for a learning algorithm that leads to better performance of the algorithm. In this project, we decided to tune the hyperparameters of the Random Forest [26] algorithm as there are a lot of parametric values that can affect the algorithm's performance. However, the parameters are not critical enough to need tuning for other algorithms, like Logistic Regression [24] and Naive Bayes [25].

Parameter optimization has always been a process that comes at the end aiming to fine tune the model to get better results than what we have already received. Random

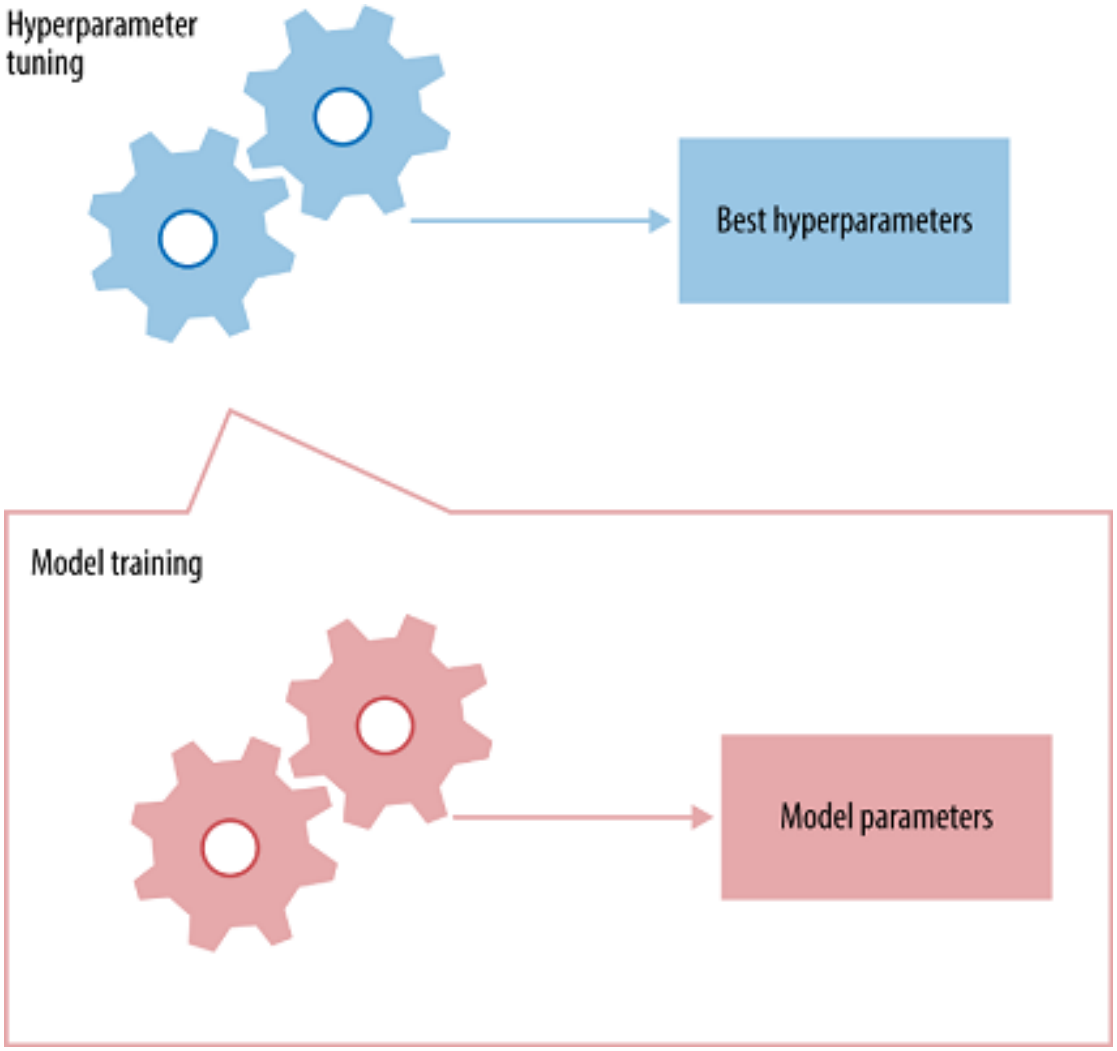

Figure 3.8: Parameter Tuning

forest has a lot of parameters that individually affect the model and therefore it becomes important to manage those parameters in a way such that through their combination we can get an optimal setting that can lead to a more robust and accurate model.

This paper uses Randomized Search [30] to optimize hyperparameters by randomly selecting points assigned inside a domain. Following this, Grid Search [30] is applied to fine-tune the hyperparameters found by forming a grid of values based on the best values provided by the Randomized Search [30].

3.8 Feature Importance

Feature Importance is the technique of calculating a score for all the input features used in the model. This score refers to the importance of the variable in predicting the target variable.

We use the built-in Feature Importance technique in the Random Forest algorithm implemented in scikit-learn using Gini Importance [31]. Gini Importance [31] averages the decrease in node impurity for each feature over all trees in the ensemble to get the feature importance.

We also use the SHAP [32] interpretation to find the feature importances from the different models. In order to get this estimate of feature contribution to the prediction, it uses the Shapley [32] from game theory which helps us fairly distribute the contribution among the features. We apply both the methods to all the final models with the self-created dense features as well as the probability estimates from the text models. To be clear, SHAP (SHapley Additive exPlanations) is a game-theoretic framework for explaining any machine learning model's output. It establishes a link between optimum credit allocation and local explanations via the use of game theory's

traditional Shapley values and their associated expansions.

The different types of plots present in the python package of SHAP helps us in visualizing the importance of each variable to the outcome. Force Plots and Summary plots are two of the most common plots that help in providing information behind the model. A short snippet of how the plots look like is given below.

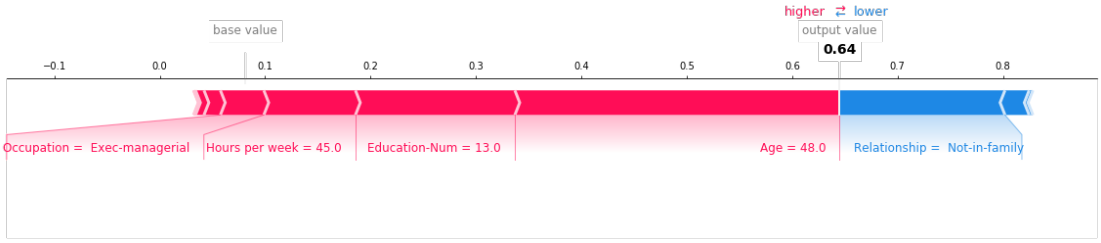

Figure 3.9: Random Snippet of Force Plot

The visualizations help in informing the contribution of each variable whether it is positively or negatively, and therefore, we can make decisions and useful inferences based on it.

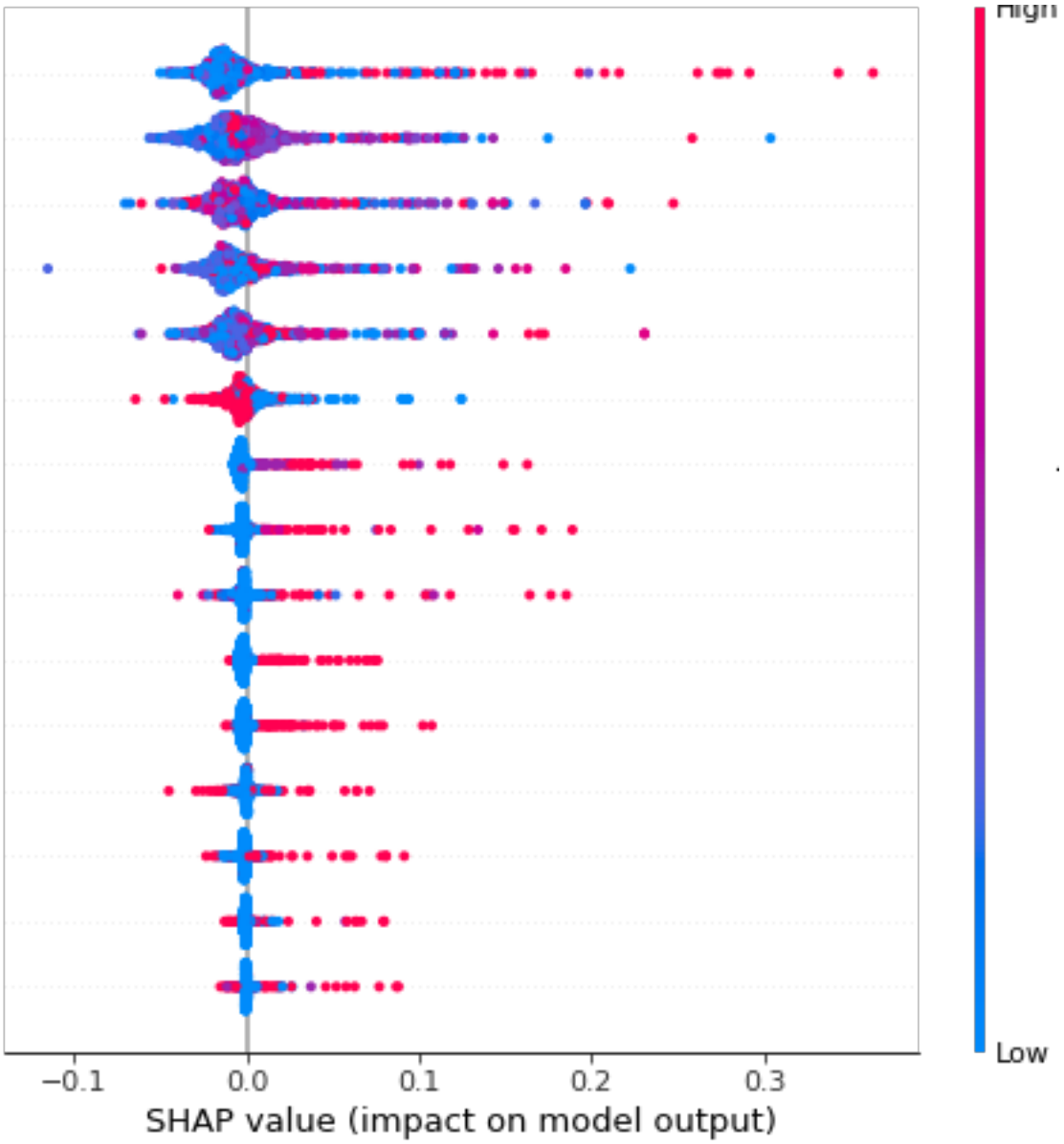

Figure 3.10: Random Snippet of Summary Plot

CHAPTER 4

RESULTS AND INFERENCE

This section is important to judge how well the research was done and to present whether the model architecture was any different than the previous works and if that difference was a good one.

There are three parts to be discussed in this section - performance of the text models, performance of the final model, and discussion of the importance of each feature in predicting the outcome. To measure the performance and understand the feature importance in the parts mentioned before, we decided to divide the dataset of 5671 into 4537 training rows and 1134 testing rows. This helps expose the classifier to train on the given data and measure its performance over the unseen data.

4.1 Model Results

For the performance of the text models, we can infer from the above plot, developed using matplotlib, that out of the three machine learning algorithms used, Random Forest [26] performs equally well for each text embedding technique, while the performance under Logistic Regression [24] improves with the use of a more complex and advanced text embedding technique. However, Gaussian Naive Bayes [25] Classifier fails to achieve a reasonable threshold with any embedding techniques. Therefore, based on the inference from the graph, we choose the Random Forest version of all the embeddings.

Now to measure the performance of the final model, the test set separated from the primary data is first converted into vectors using the different embedding methods and

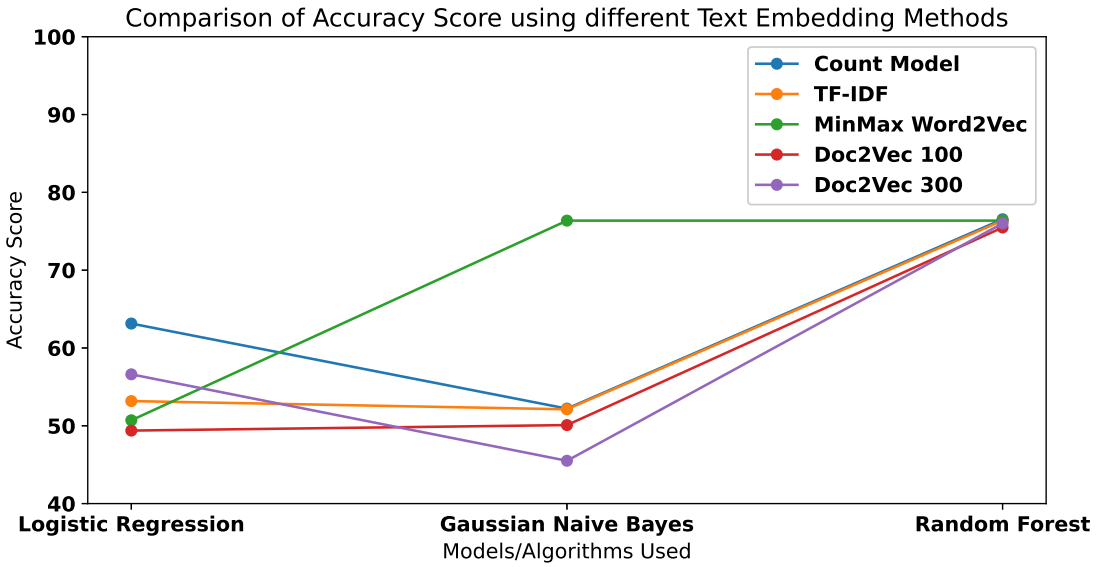

Figure 4.1: Comparison of Accuracy Score using different Text Embedding Methods

Table 4.1: Training Score using just TF-IDF Vector elements

Model	**Accuracy**
Logistic Regression	55.43
Gaussian Naive Bayes	51.4
Random Forest	74.66

then fed into the final model along with other numeric data. We make use of the metrics like Accuracy, Precision, Recall, and F-Score [29] in order to understand how the final model is performing on the test set.

The graph in Figure 4.2 shows the performance metrics of different machine learning algorithms applied over the original numeric data combined with the estimated probability received from the Random Forest versions of the embedded text data. We see that the model's accuracy remains highest at around 75 when we use Random Forest to model the final data regardless of the embedding method used. However, for my use case, we are predicting whether a user will receive a pizza or not, which is a rare outcome with a chance of 24.63%, according to my dataset. Therefore, we would also look at the measure of recall we are getting from the different models.

From the graph, we infer that both the Doc2Vec embeddings, i.e., 100- and 300-dimensions, fed into a logistic model perform much better than other models in

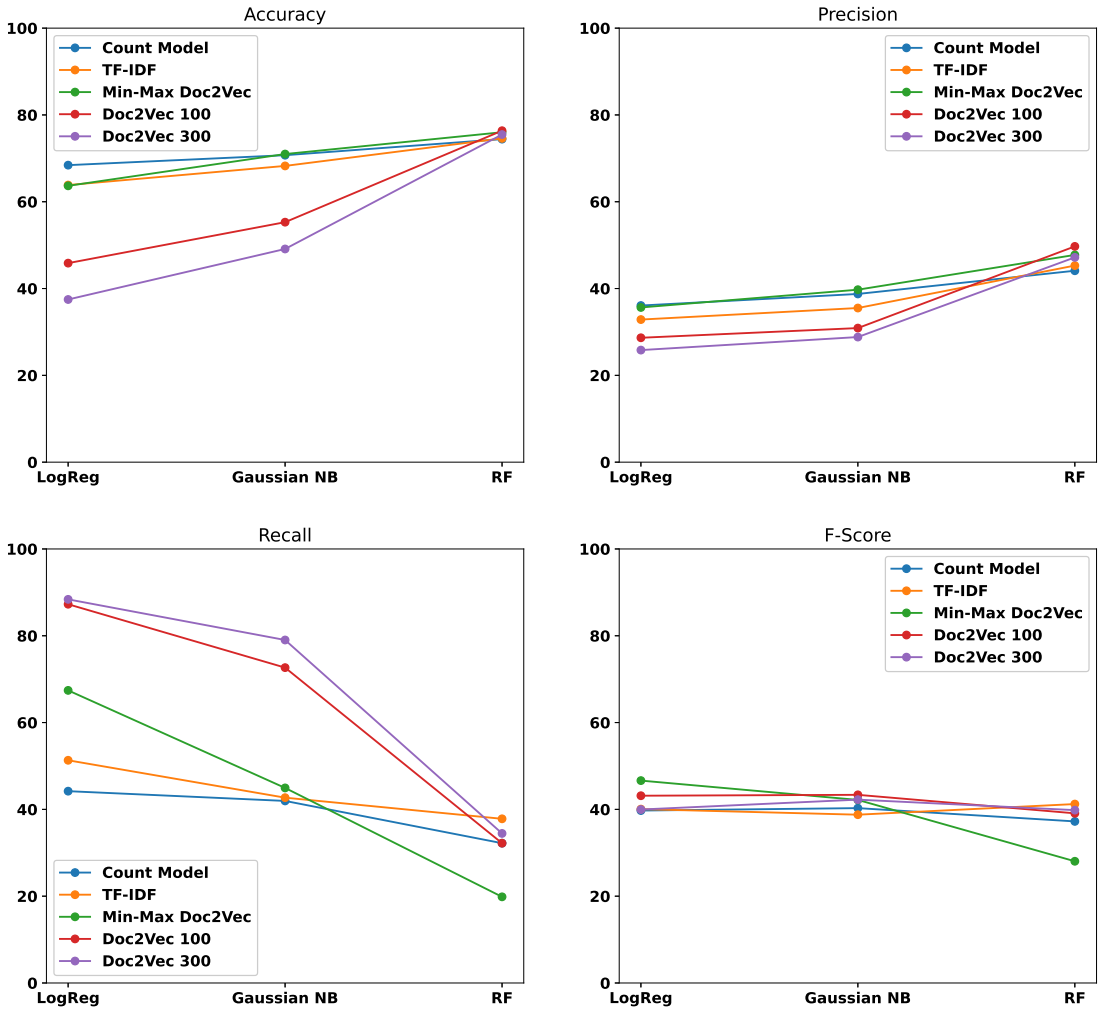

Figure 4.2: Performance of the Final Model in terms of Accuracy, Precision, Recall & F-Score

predicting the requesters that will receive a pizza. However, we also notice that the same models are less precise in predicting who actually receives the pizza and who does not.

On the other hand, the Logistic Model of the final data having Count and TF-IDF embeddings have a comparatively lower but appropriate recall score as well as a good precision score. Therefore, we attempt to get a balance between recall and precision by measuring the F-Score of the models. We see that, on average, all the models perform relatively close to each other in terms of F-score, with Logistic Regression version of Min-Max Word2Vec & TF-IDF and Random Forest version of 300-dimension of Doc2Vec & Min-Max Word2Vec have high F-scores. This suggests that the mentioned models maintain a good balance between precision and recall while predicting the outcome of a requester receiving a pizza or not.

Table 4.2: Final Accuracy Score for each combination of final model and embedding, LogReg, NB, RF - Logistic Regression, Naive Bayes, Random Forest

Models used	**Count Model**	**TF-IDF**	**Min-Max Doc2Vec**	**Doc2Vec 100**	**Doc2Vec300**
LogReg	66.49	65.78	61.02	45.14	42.41
Gaussian NB	68.95	68.43	68.25	57.05	53.26
RF	73.45	73.28	75.48	75.66	76.10

Table 4.3: Final Recall Score for each combination of final model and embedding, LogReg, NB, RF - Logistic Regression, Naive Bayes, Random Forest

Models used	**Count Model**	**TF-IDF**	**Min-Max Doc2Vec**	**Doc2Vec 100**	**Doc2Vec300**
LogReg	45.05	48.46	60.40	89.76	88.39
Gaussian NB	39.93	45.39	33.44	72.69	75.76
RF	41.97	38.90	35.49	43.34	41.97

Table 4.4: Final Precision Score for each combination of final model and embedding, LogReg, NB, RF - Logistic Regression, Naive Bayes, Random Forest

Models used	**Count Model**	**TF-IDF**	**Min-Max Doc2Vec**	**Doc2Vec 100**	**Doc2Vec300**
LogReg	37.60	37.46	35.18	30.76	29.49
Gaussian NB	39.93	40.18	37.26	34.35	32.59
RF	48.42	47.89	53.88	53.58	54.91

Table 4.5: Final F-Score for each combination of final model and embedding, LogReg, NB, RF - Logistic Regression, Naive Bayes, Random Forest

Models used	Count Model	TF-IDF	Min-Max Doc2Vec	Doc2Vec 100	Doc2Vec300
LogReg	40.99	42.26	44.47	45.81	44.23
Gaussian NB	39.93	42.62	35.25	46.65	45.58
RF	44.97	42.93	42.79	47.92	47.58

4.2 Model Inference

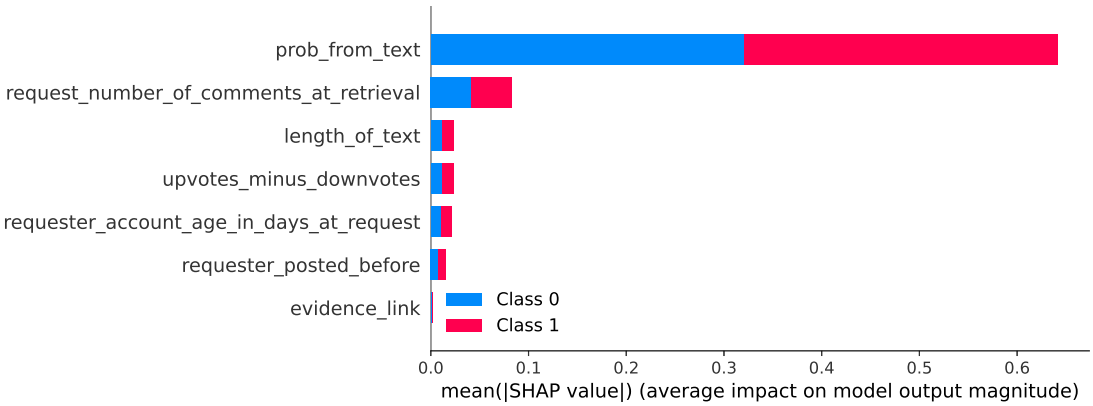

Figure 4.3: Ranking of Feature Importance in terms of SHAP values

In order to measure the importance of each feature, we use the SHAP [32] values from the classifiers. The above graph provides the mean SHAP [32] values of each feature utilized to predict the target variable. This mean-SHAP [32] value refers to the impact of a feature on the model. Therefore, we see that the probability estimate generated from the text has a significant role in predicting whether a requester will receive their pizza or not. The number of comments on the request post, the upvotes minus downvotes, and the length of the text also have some part to play in predicting the outcome. However, from the models, we see that it hardly matters whether the requester has posted before in the subreddit or has attached an evidence link in the text as their SHAP [32] values are very low.

CHAPTER 5

CONCLUSION

This work presents a stack of two models that separately deal with the text and numeric data. This allows us to reduce higher dimensional vectors generated from the text data to a probability estimate and retain the importance of the features present in the original dataset. The experimental findings validate the efficacy of my approach in terms of the performance metrics discussed in section 4.

Therefore, given a requester posting a request, my model can predict whether it will stimulate any altruistic behavior from the other Redditors. Inferring the model also gives us an understanding of how useful other variables in the dataset were in predicting the outcome.

CHAPTER 6

FUTURE ENHANCEMENT

In order to improve my work in the future, we can work around framing and combining different embedding techniques to get a vector representation of text, leading to better probability estimates and more satisfactory performance. Finally, this work can also be extended by using more sophisticated machine learning and deep learning models as well as adding more data in order to develop a more robust model.

REFERENCES

[1] Hutchins, W.J. (2004). The Georgetown-IBM Experiment Demonstrated in January 1954. In: Frederking, R.E., Taylor, K.B. (eds) Machine Translation: From Real Users to Research. AMTA 2004.

[2] Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cambria (2017). Recent Trends in Deep Learning Based Natural Language Processing. arXiv:1708.02709.

[3] Bamman, D., Doğruöz, A. S., Eisenstein, J., Hovy, D., Jurgens, D., O'Connor, B., ... Volkova (2016). Proceedings of the First Workshop on NLP and Computational Social Science.

[4] Althoff, T., Salehi, N., Nguyen, T. (2013). Random Acts of Pizza : Success Factors of Online Requests.

[5] Althoff, T., Salehi, N., Nguyen, T. (2013). Random Acts of Pizza : Success Factors of Online Requests.

[6] J. Filipczuk, E. Pesce & S. Senatore, Sentiment detection for predicting altruistic behaviors in Social Web: A case study, 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2016.

[7] Hsieh, Hsun-Ping, Yan, Rui, Li, Cheng-Te. (2016). Will I Win Your Favor? Predicting the Success of Altruistic Requests. Pacific-Asia Conference on Knowledge Discovery and Data Mining.

[8] Ahmad, Amreen & Ahmad, Tanvir & Bhatt, Abhishek. (2019). A Novel Approach for Predicting the Outcome of Request in RAOP Dataset. Proceedings of GUCON 2018.

[9] Esin Durmus & Claire Cardie. 2018. Exploring the Role of Prior Beliefs for Argument Persuasion. In Proceedings of the 2018 Conference of the North American.

[10] Diyi Yang, Jiaao Chen, Zichao Yang, Dan Jurafsky, and Eduard Hovy. 2019. Let's Make Your Request More Persuasive: Modeling Persuasive Strategies via Semi-Supervised Neural Nets on Crowdfunding Platforms. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.

[11] Chen, J., & Yang, D. (2021). Weakly-Supervised Hierarchical Models for Predicting Persuasive Strategies in Good-faith Textual Requests. AAAI.

[12] Raschka, S. (2018). Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. CoRR, arXiv:1811.12808.

[13] Loper, Edward & Bird, Steven. (2002). NLTK: the Natural Language Toolkit. CoRR, arXiv:cs/0205028

[14] Erwig, Martin & Gopinath, Rahul. (2012). Explanations for Regular Expressions.

[15] Fellbaum, C.D. (2000). WordNet : an electronic lexical database. Language, 76, 706.

[16] Harris, David & Harris, Sarah. (2007). Digital Design and Computer Architecture.

[17] Manning, C., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge: Cambridge University Press.

[18] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). "Efficient Estimation of Word Representations in Vector Space." arXiv:1301.3781.

[19] Le, Q. V., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents.

[20] H. P. Luhn, A Statistical Approach to Mechanized Encoding and Searching of Literary Information, in IBM Journal of Research and Development.

[21] SPARCK JONES, K. (1972), A Statistical Interpretation of Term Specificity and its Application in Retrieval, Journal of Documentation.

[22] Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana (2020). Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems.

[23] Boom, C. D., Van Canneyt, S., Demeester, T., & Dhoedt, B. (2016). “Representation learning for very short texts using weighted word embedding aggregation.” Pattern Recognition Letters, 80, 150–156.

[24] Cramer, J.S., The Origins of Logistic Regression (December 2002). Tinbergen Institute Working Paper No. 2002-119/4

[25] Rish, Irina. (2001). An Empirical Study of the Naïve Bayes Classifier. IJCAI 2001 Work Empir Methods Artif Intell. 3.

[26] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001).

[27] Cunningham, Padraig & Delany, Sarah. (2007). k-Nearest neighbour classifiers. Mult Classif Syst.

[28] Evgeniou, Theodoros & Pontil, Massimiliano. (2001). Support Vector Machines: Theory and Applications.

[29] Goutte, Cyril & Gaussier, Eric. (2005). A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. Lecture Notes in Computer Science.

[30] James Bergstra, Yoshua Bengio. (2012). Random Search for Hyper-Parameter Optimization.

[31] Menze, Bjoern & Kelm, Bernd & Masuch, Ralf & Himmelreich, Uwe & Bachert, Peter & Petrich, Wolfgang & Hamprecht, Fred. (2009). A comparison of Random Forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data.

[32] Lundberg, S., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. arXiv:1705.07874

APPENDIX A

CODE

```
PATH = '/Users/siddhantthakur/pizza_req/'


# importing all the necessary libraries
from tqdm import tqdm
import json
import pandas as pd
import numpy as np
from statistics import mean
import shap

from prettytable import PrettyTable
import matplotlib.pyplot as plt

import nltk
from nltk.tokenize import regexp_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
import re

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.svm import SVC

from gensim.models import KeyedVectors
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
from sklearn.model_selection import train_test_split, StratifiedKFold
    , RandomizedSearchCV, GridSearchCV
```

```
from sklearn.metrics import confusion_matrix, accuracy_score,
    f1_score, precision_score, recall_score, roc_auc_score

import warnings
warnings.filterwarnings("ignore")


# setting a random state to keep the output consistent
RANDOM_STATE = 4


nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')


# loading the json data
with open(PATH + 'pizza_request_dataset/pizza_request_dataset.json',
    'r') as f:
    data = json.load(f)


# getting the success rate of receiving a pizza
success = [d['requester_received_pizza'] for d in data]
print("Average Success Rate",str(sum(success)/len(success)*100))


# creating features like upvotes-downvotes
for d in data:
    d['upvotes_minus_downvotes'] = d['
        number_of_upvotes_of_request_at_retrieval'] - d['
        number_of_downvotes_of_request_at_retrieval']

# combining the title and textual part of the request
for d in data:
    d['final_request_text'] = d['request_title'] + " " + d['
        request_text_edit_aware']
```

```
# converting json to dataframe
df = pd.json_normalize(data)
df['requester_received_pizza'] = df['requester_received_pizza'].
    astype(int)
df = df.sample(frac=1).reset_index(drop=True)
df.head()


# printing the null values in the dataset
print(df['requester_received_pizza'].isna().sum())


# text preprocessing
stopwords_eng = stopwords.words('english')
lemmatizer = WordNetLemmatizer()

def process_text(text):
    text = text.replace("\n"," ").replace("\r"," ")
    text = re.sub(r'    ', " '' ", text)
    text = re.sub(r'    ', " '' ", text)
    text = re.sub(r'"', " '' ", text)
    text = re.sub(r'(\S)(\'\')', r'\1 \2 ', text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)

    punc_list = '!"#$%()*+,-./:;<=>?@^_{|}~[]'
    t = str.maketrans(dict.fromkeys(punc_list," "))
    text = text.translate(t)

    t = str.maketrans(dict.fromkeys("'`",""))
    text = text.translate(t)

    text = text.lower()
```

```
    tokens = regexp_tokenize(text,pattern='\s+',gaps=True)
    cleaned_tokens = []

    for t in tokens:
        if t not in stopwords_eng:
            l = lemmatizer.lemmatize(t)
            cleaned_tokens.append(l)

    return cleaned_tokens


# 20% split for train and test data
train_set = df.loc[:4536,:]
test_set = df.loc[4537:,:].reset_index(drop=True)
print(train_set.shape)
print(test_set.shape)


# In[15]:

# engineering more features using past literature
final_df = df.loc[:,['upvotes_minus_downvotes','
    requester_account_age_in_days_at_request','
    request_number_of_comments_at_retrieval']]
final_df['length_of_text'] = df.apply(lambda r: len(r['
    final_request_text']), axis = 1)
final_df['evidence_link'] = df.apply(lambda r: 1 if re.findall(r'(?:
    http\:|https\:)?\/\/.*\.',r['final_request_text']) else 0, axis =
    1)
final_df['requester_posted_before'] = df.apply(lambda r: 0 if r['
    requester_days_since_first_post_on_raop_at_request']==0 else 1,
    axis = 1)
final_df['requester_received_pizza'] = df.loc[:,'
    requester_received_pizza']


# splitting the final train and test set
final_train_set = final_df.loc[:4536,:]
```

```
final_test_set = final_df.loc[4537:,:].reset_index(drop=True)
print(final_train_set.shape)
print(final_test_set.shape)


# Random Forest based Count Model

# vectorizing the text data using WordCount
new_count_vec = CountVectorizer(analyzer=process_text)
new_count_df = new_count_vec.fit_transform(train_set['
    final_request_text'])
new_count_labels = train_set['requester_received_pizza']

print(new_count_df.shape)

# MODELING TO GET ESTIMATES AND USING
# STRATIFIED CROSS-VALIDATION AS THE DATASET IS IMBALANCED
new_skfold_tfidf = StratifiedKFold(n_splits = 10, shuffle=True,
    random_state=RANDOM_STATE)
new_count_model_lr = LogisticRegression(class_weight = {0:1, 1:5},
    random_state=RANDOM_STATE)
new_count_model_nbg = GaussianNB()
new_count_model_nbm = MultinomialNB()
new_count_model_rf = RandomForestClassifier(n_estimators=590,
    min_samples_leaf=2, min_samples_split=2, max_features='sqrt',
    max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_count_model_lr_accuracy_score = []
new_count_model_nbg_accuracy_score = []
new_count_model_nbm_accuracy_score = []
new_count_model_rf_accuracy_score = []


for train_index, test_index in new_skfold_tfidf.split(new_count_df,
    new_count_labels):
    X_train, X_test = new_count_df[train_index], new_count_df[
        test_index]
    y_train, y_test = new_count_labels[train_index], new_count_labels
        [test_index]
```

```
    new_count_model_lr.fit(X_train, y_train)
    new_count_model_lr_accuracy_score.append(accuracy_score(y_test,
        new_count_model_lr.predict(X_test)))

    new_count_model_nbg.fit(X_train.toarray(), y_train)
    new_count_model_nbg_accuracy_score.append(accuracy_score(y_test,
        new_count_model_nbg.predict(X_test.toarray())))

    new_count_model_nbm.fit(X_train, y_train)
    new_count_model_nbm_accuracy_score.append(accuracy_score(y_test,
        new_count_model_nbm.predict(X_test)))

    new_count_model_rf.fit(X_train, y_train)
    new_count_model_rf_accuracy_score.append(accuracy_score(y_test,
        new_count_model_rf.predict(X_test)))


# getting the probability estimates from text
# contributing towards the success of the request
new_final_count_rf_df = final_train_set.copy()
new_count_rf_est_prob = new_count_model_rf.predict_proba(new_count_df
    )[:,1]
new_final_count_rf_df.insert(5,'prob_from_text',new_count_rf_est_prob
    )
new_final_count_rf_df

# finalising the training data for the second model
new_final_count_rf_df = new_final_count_rf_df.sample(frac=1).
    reset_index(drop=True)
new_final_count_rf_labels = new_final_count_rf_df.iloc[:,-1]
new_final_count_rf_df = new_final_count_rf_df.iloc[:,:-1]
new_final_count_rf_labels


# MODELING TO GET ESTIMATES AND USING
# STRATIFIED CROSS-VALIDATION AS THE DATASET IS IMBALANCED
new_skfold_final_count_rf = StratifiedKFold(n_splits = 10, shuffle=
    True, random_state=RANDOM_STATE)
```

```
new_final_count_rf_model_lr = LogisticRegression(class_weight = {0:1,
    1:5}, random_state=RANDOM_STATE)
new_final_count_rf_model_nbg = GaussianNB()
new_final_count_rf_model_rf = RandomForestClassifier(n_estimators
   =590, min_samples_leaf=2, min_samples_split=2, max_features='sqrt'
   ,max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_final_count_rf_model_lr_accuracy_score = []
new_final_count_rf_model_nbg_accuracy_score = []
new_final_count_rf_model_rf_accuracy_score = []

new_final_count_rf_model_lr_precision_score = []
new_final_count_rf_model_nbg_precision_score = []
new_final_count_rf_model_rf_precision_score = []

new_final_count_rf_model_lr_recall_score = []
new_final_count_rf_model_nbg_recall_score = []
new_final_count_rf_model_rf_recall_score = []

new_final_count_rf_model_lr_f1_score = []
new_final_count_rf_model_nbg_f1_score = []
new_final_count_rf_model_rf_f1_score = []


for train_index, test_index in new_skfold_final_count_rf.split(
   new_final_count_rf_df, new_final_count_rf_labels):
    X_train, X_test = new_final_count_rf_df.iloc[train_index,:],
        new_final_count_rf_df.iloc[test_index,:]
    y_train, y_test = new_final_count_rf_labels[train_index],
        new_final_count_rf_labels[test_index]

    new_final_count_rf_model_lr.fit(X_train,y_train)
    new_final_count_rf_model_lr_accuracy_score.append(accuracy_score(
        y_test,new_final_count_rf_model_lr.predict(X_test)))
    new_final_count_rf_model_lr_precision_score.append(
        precision_score(y_test,new_final_count_rf_model_lr.predict(
        X_test)))
    new_final_count_rf_model_lr_recall_score.append(recall_score(
        y_test,new_final_count_rf_model_lr.predict(X_test)))
```

```
    new_final_count_rf_model_lr_f1_score.append(f1_score(y_test,
        new_final_count_rf_model_lr.predict(X_test)))

    new_final_count_rf_model_nbg.fit(X_train,y_train)
    new_final_count_rf_model_nbg_accuracy_score.append(accuracy_score
        (y_test,new_final_count_rf_model_nbg.predict(X_test)))
    new_final_count_rf_model_nbg_precision_score.append(
        precision_score(y_test,new_final_count_rf_model_nbg.predict(
        X_test)))
    new_final_count_rf_model_nbg_recall_score.append(recall_score(
        y_test,new_final_count_rf_model_nbg.predict(X_test)))
    new_final_count_rf_model_nbg_f1_score.append(f1_score(y_test,
        new_final_count_rf_model_nbg.predict(X_test)))

    new_final_count_rf_model_rf.fit(X_train,y_train)
    new_final_count_rf_model_rf_accuracy_score.append(accuracy_score(
        y_test,new_final_count_rf_model_rf.predict(X_test)))
    new_final_count_rf_model_rf_precision_score.append(
        precision_score(y_test,new_final_count_rf_model_rf.predict(
        X_test)))
    new_final_count_rf_model_rf_recall_score.append(recall_score(
        y_test,new_final_count_rf_model_rf.predict(X_test)))
    new_final_count_rf_model_rf_f1_score.append(f1_score(y_test,
        new_final_count_rf_model_rf.predict(X_test)))


# using shapley values to understand importance of each feature
count_explainer = shap.TreeExplainer(new_final_count_rf_model_rf)
count_shap_values = count_explainer.shap_values(X_test)
shap.summary_plot(count_shap_values, X_test, plot_type="bar")
plt.savefig('eps/FeatureImportanceShapley_count.eps',dpi=1200,
    bbox_inches='tight')

# using the feature importance feature of
# the Random Forest model that uses Gini Importance
sorted_idx = new_final_count_rf_model_rf.feature_importances_.argsort
    ()
plt.barh(new_final_count_rf_df.columns[sorted_idx],
    new_final_count_rf_model_rf.feature_importances_[sorted_idx])
```

```
# working with the test data and
# transforming the test text data
test_count_df = new_count_vec.transform(test_set['final_request_text'
    ])
print(test_count_df.shape)

# getting the probability estimate from the test text data
# from the first model to be used in the second one
test_final_count_rf_df = final_test_set.copy()
test_count_rf_est_prob = new_count_model_rf.predict_proba(
    test_count_df)[:,1]
test_final_count_rf_df.insert(5,'prob_from_text',
    test_count_rf_est_prob)
test_final_count_rf_df


test_final_count_rf_df = test_final_count_rf_df.sample(frac=1).
    reset_index(drop=True)
test_final_count_rf_labels = test_final_count_rf_df.iloc[:,-1]
test_final_count_rf_df = test_final_count_rf_df.iloc[:,:-1]
test_final_count_rf_labels



# Random Forest based TF-IDF Model

new_tfidf_vec = TfidfVectorizer(analyzer=process_text)
new_tfidf_df = new_tfidf_vec.fit_transform(train_set['
    final_request_text'])

new_tfidf_labels = train_set['requester_received_pizza']

print(new_tfidf_df.shape)


new_skfold_tfidf = StratifiedKFold(n_splits = 10, shuffle=True,
    random_state=RANDOM_STATE)
```

```
new_tfidf_model_lr = LogisticRegression(class_weight = {0:1, 1:5},
    random_state=RANDOM_STATE)
new_tfidf_model_nbg = GaussianNB()
new_tfidf_model_nbm = MultinomialNB()
new_tfidf_model_rf = RandomForestClassifier(n_estimators=590,
    min_samples_leaf=2, min_samples_split=2, max_features='sqrt',
    max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_tfidf_model_lr_accuracy_score = []
new_tfidf_model_nbg_accuracy_score = []
new_tfidf_model_nbm_accuracy_score = []
new_tfidf_model_rf_accuracy_score = []


for train_index, test_index in new_skfold_tfidf.split(new_tfidf_df,
    new_tfidf_labels):
    X_train, X_test = new_tfidf_df[train_index], new_tfidf_df[
        test_index]
    y_train, y_test = new_tfidf_labels[train_index], new_tfidf_labels
        [test_index]

    new_tfidf_model_lr.fit(X_train,y_train)
    new_tfidf_model_lr_accuracy_score.append(accuracy_score(y_test,
        new_tfidf_model_lr.predict(X_test)))

    new_tfidf_model_nbg.fit(X_train.toarray(),y_train)
    new_tfidf_model_nbg_accuracy_score.append(accuracy_score(y_test,
        new_tfidf_model_nbg.predict(X_test.toarray())))

    new_tfidf_model_nbm.fit(X_train,y_train)
    new_tfidf_model_nbm_accuracy_score.append(accuracy_score(y_test,
        new_tfidf_model_nbm.predict(X_test)))

    new_tfidf_model_rf.fit(X_train,y_train)
    new_tfidf_model_rf_accuracy_score.append(accuracy_score(y_test,
        new_tfidf_model_rf.predict(X_test)))


new_final_tfidf_rf_df = final_train_set.copy()
```

```
new_tfidf_rf_est_prob = new_tfidf_model_rf.predict_proba(new_tfidf_df
    )[:,1]
new_final_tfidf_rf_df.insert(5,'prob_from_text',new_tfidf_rf_est_prob
    )
new_final_tfidf_rf_df


new_final_tfidf_rf_df = new_final_tfidf_rf_df.sample(frac=1).
    reset_index(drop=True)
new_final_tfidf_rf_labels = new_final_tfidf_rf_df.iloc[:,-1]
new_final_tfidf_rf_df = new_final_tfidf_rf_df.iloc[:,:-1]
new_final_tfidf_rf_labels


new_skfold_final_tfidf_rf = StratifiedKFold(n_splits = 10, shuffle=
    True, random_state=RANDOM_STATE)
new_final_tfidf_rf_model_lr = LogisticRegression(class_weight = {0:1,
     1:5}, random_state=RANDOM_STATE)
new_final_tfidf_rf_model_nbg = GaussianNB()
new_final_tfidf_rf_model_rf = RandomForestClassifier(n_estimators
    =590, min_samples_leaf=2, min_samples_split=2, max_features='sqrt'
    ,max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_final_tfidf_rf_model_lr_accuracy_score = []
new_final_tfidf_rf_model_nbg_accuracy_score = []
new_final_tfidf_rf_model_rf_accuracy_score = []

new_final_tfidf_rf_model_lr_precision_score = []
new_final_tfidf_rf_model_nbg_precision_score = []
new_final_tfidf_rf_model_rf_precision_score = []

new_final_tfidf_rf_model_lr_recall_score = []
new_final_tfidf_rf_model_nbg_recall_score = []
new_final_tfidf_rf_model_rf_recall_score = []

new_final_tfidf_rf_model_lr_f1_score = []
new_final_tfidf_rf_model_nbg_f1_score = []
new_final_tfidf_rf_model_rf_f1_score = []
```

```
for train_index, test_index in new_skfold_final_tfidf_rf.split(
    new_final_tfidf_rf_df, new_final_tfidf_rf_labels):
    X_train, X_test = new_final_tfidf_rf_df.iloc[train_index,:],
        new_final_tfidf_rf_df.iloc[test_index,:]
    y_train, y_test = new_final_tfidf_rf_labels[train_index],
        new_final_tfidf_rf_labels[test_index]

    new_final_tfidf_rf_model_lr.fit(X_train,y_train)
    new_final_tfidf_rf_model_lr_accuracy_score.append(accuracy_score(
        y_test,new_final_tfidf_rf_model_lr.predict(X_test)))
    new_final_tfidf_rf_model_lr_precision_score.append(
        precision_score(y_test,new_final_tfidf_rf_model_lr.predict(
        X_test)))
    new_final_tfidf_rf_model_lr_recall_score.append(recall_score(
        y_test,new_final_tfidf_rf_model_lr.predict(X_test)))
    new_final_tfidf_rf_model_lr_f1_score.append(f1_score(y_test,
        new_final_tfidf_rf_model_lr.predict(X_test)))

    new_final_tfidf_rf_model_nbg.fit(X_train,y_train)
    new_final_tfidf_rf_model_nbg_accuracy_score.append(accuracy_score
        (y_test,new_final_tfidf_rf_model_nbg.predict(X_test)))
    new_final_tfidf_rf_model_nbg_precision_score.append(
        precision_score(y_test,new_final_tfidf_rf_model_nbg.predict(
        X_test)))
    new_final_tfidf_rf_model_nbg_recall_score.append(recall_score(
        y_test,new_final_tfidf_rf_model_nbg.predict(X_test)))
    new_final_tfidf_rf_model_nbg_f1_score.append(f1_score(y_test,
        new_final_tfidf_rf_model_nbg.predict(X_test)))

    new_final_tfidf_rf_model_rf.fit(X_train,y_train)
    new_final_tfidf_rf_model_rf_accuracy_score.append(accuracy_score(
        y_test,new_final_tfidf_rf_model_rf.predict(X_test)))
    new_final_tfidf_rf_model_rf_precision_score.append(
        precision_score(y_test,new_final_tfidf_rf_model_rf.predict(
        X_test)))
    new_final_tfidf_rf_model_rf_recall_score.append(recall_score(
        y_test,new_final_tfidf_rf_model_rf.predict(X_test)))
```

```
    new_final_tfidf_rf_model_rf_f1_score.append(f1_score(y_test,
        new_final_tfidf_rf_model_rf.predict(X_test)))


# using shapley values to understand importance of each feature
tfidf_explainer = shap.TreeExplainer(new_final_tfidf_rf_model_rf)
tfidf_shap_values = tfidf_explainer.shap_values(X_test)
shap.summary_plot(tfidf_shap_values, X_test, plot_type="bar")
plt.savefig('eps/FeatureImportanceShapley_rf.eps',dpi=1200,
    bbox_inches='tight')


sorted_idx = new_final_tfidf_rf_model_rf.feature_importances_.argsort
    ()
plt.barh(new_final_tfidf_rf_df.columns[sorted_idx],
    new_final_tfidf_rf_model_rf.feature_importances_[sorted_idx])


test_tfidf_df = new_tfidf_vec.transform(test_set['final_request_text'
    ])
print(test_tfidf_df.shape)


test_final_tfidf_rf_df = final_test_set.copy()
test_tfidf_rf_est_prob = new_tfidf_model_rf.predict_proba(
    test_tfidf_df)[:,1]
test_final_tfidf_rf_df.insert(5,'prob_from_text',
    test_tfidf_rf_est_prob)
test_final_tfidf_rf_df


test_final_tfidf_rf_df = test_final_tfidf_rf_df.sample(frac=1).
    reset_index(drop=True)
test_final_tfidf_rf_labels = test_final_tfidf_rf_df.iloc[:,-1]
test_final_tfidf_rf_df = test_final_tfidf_rf_df.iloc[:,:-1]
test_final_tfidf_rf_labels


# Random Forest based MinMax Word2Vec Model
```

```
# loading the Google News Vectors model
model = KeyedVectors.load_word2vec_format(PATH + "GoogleNews-vectors-
    negative300.bin", limit = 10**6,binary=True)

# getting the vector representation for each word in the loaded model
def get_vec(word):
    try:
        return model[word]
    except:
        return np.zeros(300)

# getting the minimum and maximum of the vector representation from
    the text
# and concatenating them to form a double vector representation
def get_sent_vec_min_max(sent):
    tokens = process_text(sent)
    min_vector = np.min([get_vec(t) for t in tokens], axis = 0)
    max_vector = np.max([get_vec(t) for t in tokens], axis = 0)
    vector = np.concatenate([min_vector,max_vector])

    return vector

new_minmax_word2vec_emb = train_set.apply(lambda r:
    get_sent_vec_min_max(r['final_request_text']), axis=1)
new_minmax_v = np.column_stack([new_minmax_word2vec_emb.values.tolist
    ()])
new_minmax_word2vec_df = pd.DataFrame(new_minmax_v)
new_minmax_word2vec_labels = train_set['requester_received_pizza']
new_minmax_word2vec_df



new_skfold_minmax_word2vec = StratifiedKFold(n_splits = 10, shuffle=
    True, random_state=RANDOM_STATE)
new_minmax_word2vec_model_lr = LogisticRegression(class_weight =
    {0:1, 1:5}, random_state=RANDOM_STATE)
new_minmax_word2vec_model_nbg = GaussianNB()
new_minmax_word2vec_model_rf = RandomForestClassifier(n_estimators
    =590, min_samples_leaf=2, min_samples_split=2, max_features='sqrt'
```

```
    ,max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_minmax_word2vec_model_lr_accuracy_score = []
new_minmax_word2vec_model_nbg_accuracy_score = []
new_minmax_word2vec_model_rf_accuracy_score = []

for train_index, test_index in new_skfold_minmax_word2vec.split(
    new_minmax_word2vec_df, new_minmax_word2vec_labels):
    X_train, X_test = new_minmax_word2vec_df.iloc[train_index,:],
        new_minmax_word2vec_df.iloc[test_index,:]
    y_train, y_test = new_minmax_word2vec_labels[train_index],
        new_minmax_word2vec_labels[test_index]

    new_minmax_word2vec_model_lr.fit(X_train,y_train)
    new_minmax_word2vec_model_lr_accuracy_score.append(accuracy_score
        (y_test,new_minmax_word2vec_model_lr.predict(X_test)))

    new_minmax_word2vec_model_nbg.fit(X_train,y_train)
    new_minmax_word2vec_model_nbg_accuracy_score.append(
        accuracy_score(y_test,new_minmax_word2vec_model_nbg.predict(
        X_test)))

    new_minmax_word2vec_model_rf.fit(X_train,y_train)
    new_minmax_word2vec_model_rf_accuracy_score.append(accuracy_score
        (y_test,new_minmax_word2vec_model_rf.predict(X_test)))


new_final_minmax_word2vec_rf_df = final_train_set.copy()
new_minmax_word2vec_rf_est_prob = new_minmax_word2vec_model_rf.
    predict_proba(new_minmax_word2vec_df)[:,1]
new_final_minmax_word2vec_rf_df.insert(5,'prob_from_text',
    new_minmax_word2vec_rf_est_prob)


new_final_minmax_word2vec_rf_df = new_final_minmax_word2vec_rf_df.
    sample(frac=1).reset_index(drop=True)
new_final_minmax_word2vec_rf_labels = new_final_minmax_word2vec_rf_df
    .iloc[:,-1]
```

```
new_final_minmax_word2vec_rf_df = new_final_minmax_word2vec_rf_df.
    iloc[:,:-1]
new_final_minmax_word2vec_rf_labels


new_skfold_final_minmax_word2vec = StratifiedKFold(n_splits = 10,
    shuffle=True, random_state=RANDOM_STATE)
new_final_minmax_word2vec_model_lr = LogisticRegression(class_weight
    = {0:1, 1:5}, random_state=RANDOM_STATE)
new_final_minmax_word2vec_model_nbg = GaussianNB()
new_final_minmax_word2vec_model_rf = RandomForestClassifier(
    n_estimators=590, min_samples_leaf=2, min_samples_split=2,
    max_features='sqrt',max_depth=50, bootstrap=False, random_state=
    RANDOM_STATE)

new_final_minmax_word2vec_model_lr_accuracy_score = []
new_final_minmax_word2vec_model_nbg_accuracy_score = []
new_final_minmax_word2vec_model_rf_accuracy_score = []

new_final_minmax_word2vec_model_lr_precision_score = []
new_final_minmax_word2vec_model_nbg_precision_score = []
new_final_minmax_word2vec_model_rf_precision_score = []

new_final_minmax_word2vec_model_lr_recall_score = []
new_final_minmax_word2vec_model_nbg_recall_score = []
new_final_minmax_word2vec_model_rf_recall_score = []

new_final_minmax_word2vec_model_lr_f1_score = []
new_final_minmax_word2vec_model_nbg_f1_score = []
new_final_minmax_word2vec_model_rf_f1_score = []


for train_index, test_index in new_skfold_final_minmax_word2vec.split
    (new_final_minmax_word2vec_rf_df,
    new_final_minmax_word2vec_rf_labels):
     X_train, X_test = new_final_minmax_word2vec_rf_df.iloc[
         train_index,:], new_final_minmax_word2vec_rf_df.iloc[
         test_index,:]
```

```
y_train , y_test = new_final_minmax_word2vec_rf_labels[train_index
    ], new_final_minmax_word2vec_rf_labels[test_index]

new_final_minmax_word2vec_model_lr.fit(X_train , y_train)
new_final_minmax_word2vec_model_lr_accuracy_score.append(
    accuracy_score(y_test , new_final_minmax_word2vec_model_lr.
    predict(X_test)))
new_final_minmax_word2vec_model_lr_precision_score.append(
    precision_score(y_test , new_final_minmax_word2vec_model_lr.
    predict(X_test)))
new_final_minmax_word2vec_model_lr_recall_score.append(
    recall_score(y_test , new_final_minmax_word2vec_model_lr.predict
    (X_test)))
new_final_minmax_word2vec_model_lr_f1_score.append(f1_score(
    y_test , new_final_minmax_word2vec_model_lr.predict(X_test)))

new_final_minmax_word2vec_model_nbg.fit(X_train , y_train)
new_final_minmax_word2vec_model_nbg_accuracy_score.append(
    accuracy_score(y_test , new_final_minmax_word2vec_model_nbg.
    predict(X_test)))
new_final_minmax_word2vec_model_nbg_precision_score.append(
    precision_score(y_test , new_final_minmax_word2vec_model_nbg.
    predict(X_test)))
new_final_minmax_word2vec_model_nbg_recall_score.append(
    recall_score(y_test , new_final_minmax_word2vec_model_nbg.
    predict(X_test)))
new_final_minmax_word2vec_model_nbg_f1_score.append(f1_score(
    y_test , new_final_minmax_word2vec_model_nbg.predict(X_test)))

new_final_minmax_word2vec_model_rf.fit(X_train , y_train)
new_final_minmax_word2vec_model_rf_accuracy_score.append(
    accuracy_score(y_test , new_final_minmax_word2vec_model_rf.
    predict(X_test)))
new_final_minmax_word2vec_model_rf_precision_score.append(
    precision_score(y_test , new_final_minmax_word2vec_model_rf.
    predict(X_test)))
new_final_minmax_word2vec_model_rf_recall_score.append(
    recall_score(y_test , new_final_minmax_word2vec_model_rf.predict
    (X_test)))
```

```
    new_final_minmax_word2vec_model_rf_f1_score.append(f1_score(
        y_test,new_final_minmax_word2vec_model_rf.predict(X_test)))

t = PrettyTable()
t.field_names = ['Model','Accuracy','Precision','Recall','F1-Score']
t.add_rows([
    ['Logistic Regression', mean(
        new_final_minmax_word2vec_model_lr_accuracy_score),mean(
        new_final_minmax_word2vec_model_lr_precision_score),mean(
        new_final_minmax_word2vec_model_lr_recall_score),mean(
        new_final_minmax_word2vec_model_lr_f1_score)],
    ['Gaussian NB', mean(
        new_final_minmax_word2vec_model_nbg_accuracy_score),mean(
        new_final_minmax_word2vec_model_nbg_precision_score),mean(
        new_final_minmax_word2vec_model_nbg_recall_score),mean(
        new_final_minmax_word2vec_model_nbg_f1_score)],
    ['Random Forest', mean(
        new_final_minmax_word2vec_model_rf_accuracy_score),mean(
        new_final_minmax_word2vec_model_rf_precision_score),mean(
        new_final_minmax_word2vec_model_rf_recall_score),mean(
        new_final_minmax_word2vec_model_rf_f1_score)],
])
t

# using shapley values to understand importance of each feature
minmax_word2vec_explainer = shap.TreeExplainer(
    new_final_minmax_word2vec_model_rf)
minmax_word2vec_shap_values = minmax_word2vec_explainer.shap_values(
    X_test)
shap.summary_plot(minmax_word2vec_shap_values, X_test, plot_type="bar
    ",show=False)
plt.savefig('eps/FeatureImportanceShapley_minmax.eps',dpi=1200,
    bbox_inches='tight')


sorted_idx = new_final_minmax_word2vec_model_rf.feature_importances_.
    argsort()
plt.barh(new_final_minmax_word2vec_rf_df.columns[sorted_idx],
    new_final_minmax_word2vec_model_rf.feature_importances_[sorted_idx
```

```
    ])


test_minmax_word2vec_emb = test_set.apply(lambda r:
    get_sent_vec_min_max(r['final_request_text']), axis=1)
test_minmax_v = np.column_stack([test_minmax_word2vec_emb.values.
    tolist()])
test_minmax_word2vec_df = pd.DataFrame(test_minmax_v)
test_minmax_word2vec_labels = test_set['requester_received_pizza']
test_minmax_word2vec_df


test_final_minmax_word2vec_rf_df = final_test_set.copy()
test_tfidf_minmax_word2vec_est_prob = new_minmax_word2vec_model_rf.
    predict_proba(test_minmax_word2vec_df)[:,1]
test_final_minmax_word2vec_rf_df.insert(5,'prob_from_text',
    test_tfidf_minmax_word2vec_est_prob)
test_final_minmax_word2vec_rf_df


test_final_minmax_word2vec_rf_df = test_final_minmax_word2vec_rf_df.
    sample(frac=1).reset_index(drop=True)
test_final_minmax_word2vec_rf_labels =
    test_final_minmax_word2vec_rf_df.iloc[:,-1]
test_final_minmax_word2vec_rf_df = test_final_minmax_word2vec_rf_df.
    iloc[:,:-1]
test_final_minmax_word2vec_rf_labels


# Random Forest based Doc2Vec 100 Model


new_tagged_data = [TaggedDocument(words=process_text(_d.lower()),
    tags=[str(i)]) for i, _d in enumerate(train_set['
    final_request_text'])]

new_doc2vec_100_model = Doc2Vec(vector_size=100, min_count=2, epochs
    = 50)
```

```
new_doc2vec_100_model.build_vocab(new_tagged_data)
new_doc2vec_100_model.train(new_tagged_data, epochs=
    new_doc2vec_100_model.epochs, total_examples=new_doc2vec_100_model
    .corpus_count)

new_doc2vec_100_v = np.row_stack([new_doc2vec_100_model.infer_vector(
    process_text(train_set['final_request_text'][x])) for x in range(
    len(train_set))])
new_doc2vec_100_df = pd.DataFrame(new_doc2vec_100_v)
new_doc2vec_100_labels = train_set['requester_received_pizza']



new_skfold_doc2vec_100 = StratifiedKFold(n_splits = 10, shuffle=True,
     random_state=RANDOM_STATE)
new_doc2vec_100_model_lr = LogisticRegression(class_weight = {0:1,
    1:5}, random_state=RANDOM_STATE)
new_doc2vec_100_model_nbg = GaussianNB()
new_doc2vec_100_model_rf = RandomForestClassifier(n_estimators=590,
    min_samples_leaf=2, min_samples_split=2, max_features='sqrt',
    max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_doc2vec_100_model_lr_accuracy_score = []
new_doc2vec_100_model_nbg_accuracy_score = []
new_doc2vec_100_model_rf_accuracy_score = []

for train_index, test_index in new_skfold_doc2vec_100.split(
    new_doc2vec_100_df, new_doc2vec_100_labels):
    X_train, X_test = new_doc2vec_100_df.iloc[train_index,:],
        new_doc2vec_100_df.iloc[test_index,:]
    y_train, y_test = new_doc2vec_100_labels[train_index],
        new_doc2vec_100_labels[test_index]

    new_doc2vec_100_model_lr.fit(X_train,y_train)
    new_doc2vec_100_model_lr_accuracy_score.append(accuracy_score(
        y_test,new_doc2vec_100_model_lr.predict(X_test)))

    new_doc2vec_100_model_nbg.fit(X_train,y_train)
    new_doc2vec_100_model_nbg_accuracy_score.append(accuracy_score(
        y_test,new_doc2vec_100_model_nbg.predict(X_test)))
```

```
    new_doc2vec_100_model_rf.fit(X_train,y_train)
    new_doc2vec_100_model_rf_accuracy_score.append(accuracy_score(
        y_test,new_doc2vec_100_model_rf.predict(X_test)))


new_final_doc2vec_100_rf_df = final_train_set.copy()
new_doc2vec_100_rf_est_prob = new_doc2vec_100_model_rf.predict_proba(
    new_doc2vec_100_df)[:,1]
new_final_doc2vec_100_rf_df.insert(5,'prob_from_text',
    new_doc2vec_100_rf_est_prob)

new_final_doc2vec_100_rf_df = new_final_doc2vec_100_rf_df.sample(frac
    =1).reset_index(drop=True)
new_final_doc2vec_100_rf_labels = new_final_doc2vec_100_rf_df.iloc
    [:,-1]
new_final_doc2vec_100_rf_df = new_final_doc2vec_100_rf_df.iloc[:,:-1]
new_final_doc2vec_100_rf_labels


new_skfold_final_doc2vec_100 = StratifiedKFold(n_splits = 10,shuffle=
    True, random_state=RANDOM_STATE)
new_final_doc2vec_100_model_lr = LogisticRegression(class_weight =
    {0:1, 1:5}, random_state=RANDOM_STATE)
new_final_doc2vec_100_model_nbg = GaussianNB()
new_final_doc2vec_100_model_rf = RandomForestClassifier(n_estimators
    =590, min_samples_leaf=2, min_samples_split=2, max_features='sqrt'
    ,max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_final_doc2vec_100_model_lr_accuracy_score = []
new_final_doc2vec_100_model_nbg_accuracy_score = []
new_final_doc2vec_100_model_rf_accuracy_score = []

new_final_doc2vec_100_model_lr_precision_score = []
new_final_doc2vec_100_model_nbg_precision_score = []
new_final_doc2vec_100_model_rf_precision_score = []

new_final_doc2vec_100_model_lr_recall_score = []
new_final_doc2vec_100_model_nbg_recall_score = []
```

```
new_final_doc2vec_100_model_rf_recall_score = []

new_final_doc2vec_100_model_lr_f1_score = []
new_final_doc2vec_100_model_nbg_f1_score = []
new_final_doc2vec_100_model_rf_f1_score = []


for train_index, test_index in new_skfold_final_doc2vec_100.split(
    new_final_doc2vec_100_rf_df, new_final_doc2vec_100_rf_labels):
    X_train, X_test = new_final_doc2vec_100_rf_df.iloc[train_index
        ,:], new_final_doc2vec_100_rf_df.iloc[test_index,:]
    y_train, y_test = new_final_doc2vec_100_rf_labels[train_index],
        new_final_doc2vec_100_rf_labels[test_index]

    new_final_doc2vec_100_model_lr.fit(X_train, y_train)
    new_final_doc2vec_100_model_lr_accuracy_score.append(
        accuracy_score(y_test, new_final_doc2vec_100_model_lr.predict(
        X_test)))
    new_final_doc2vec_100_model_lr_precision_score.append(
        precision_score(y_test, new_final_doc2vec_100_model_lr.predict(
        X_test)))
    new_final_doc2vec_100_model_lr_recall_score.append(recall_score(
        y_test, new_final_doc2vec_100_model_lr.predict(X_test)))
    new_final_doc2vec_100_model_lr_f1_score.append(f1_score(y_test,
        new_final_doc2vec_100_model_lr.predict(X_test)))

    new_final_doc2vec_100_model_nbg.fit(X_train, y_train)
    new_final_doc2vec_100_model_nbg_accuracy_score.append(
        accuracy_score(y_test, new_final_doc2vec_100_model_nbg.predict(
        X_test)))
    new_final_doc2vec_100_model_nbg_precision_score.append(
        precision_score(y_test, new_final_doc2vec_100_model_nbg.predict
        (X_test)))
    new_final_doc2vec_100_model_nbg_recall_score.append(recall_score(
        y_test, new_final_doc2vec_100_model_nbg.predict(X_test)))
    new_final_doc2vec_100_model_nbg_f1_score.append(f1_score(y_test,
        new_final_doc2vec_100_model_nbg.predict(X_test)))

    new_final_doc2vec_100_model_rf.fit(X_train, y_train)
```

```
    new_final_doc2vec_100_model_rf_accuracy_score.append(
        accuracy_score(y_test,new_final_doc2vec_100_model_rf.predict(
        X_test)))
    new_final_doc2vec_100_model_rf_precision_score.append(
        precision_score(y_test,new_final_doc2vec_100_model_rf.predict(
        X_test)))
    new_final_doc2vec_100_model_rf_recall_score.append(recall_score(
        y_test,new_final_doc2vec_100_model_rf.predict(X_test)))
    new_final_doc2vec_100_model_rf_f1_score.append(f1_score(y_test,
        new_final_doc2vec_100_model_rf.predict(X_test)))


# using shapley values to understand importance of each feature
doc2vec_100_explainer = shap.TreeExplainer(
    new_final_doc2vec_100_model_rf)
doc2vec_100_shap_values = doc2vec_100_explainer.shap_values(X_test)
shap.summary_plot(doc2vec_100_shap_values, X_test, plot_type="bar")
plt.savefig('eps/FeatureImportanceShapley_doc2vec_100.eps',dpi=1200,
    bbox_inches='tight')


sorted_idx = new_final_doc2vec_100_model_rf.feature_importances_.
    argsort()
plt.barh(new_final_doc2vec_100_rf_df.columns[sorted_idx],
    new_final_doc2vec_100_model_rf.feature_importances_[sorted_idx])


test_doc2vec_100_v = np.row_stack([new_doc2vec_100_model.infer_vector
    (process_text(test_set['final_request_text'][x])) for x in range(
    len(test_set))])
test_doc2vec_100_df = pd.DataFrame(test_doc2vec_100_v)


test_final_doc2vec_100_rf_df = final_test_set.copy()
test_tfidf_doc2vec_100_est_prob = new_doc2vec_100_model_rf.
    predict_proba(test_doc2vec_100_df)[:,1]
test_final_doc2vec_100_rf_df.insert(5,'prob_from_text',
    test_tfidf_doc2vec_100_est_prob)
test_final_doc2vec_100_rf_df
```

```
test_final_doc2vec_100_rf_df = test_final_doc2vec_100_rf_df.sample(
    frac=1).reset_index(drop=True)
test_final_doc2vec_100_rf_labels = test_final_doc2vec_100_rf_df.iloc
    [:,-1]
test_final_doc2vec_100_rf_df = test_final_doc2vec_100_rf_df.iloc
    [:,:-1]
test_final_doc2vec_100_rf_labels
```

```
# Random Forest based Doc2Vec 300 Model
```

```
new_tagged_data = [TaggedDocument(words=process_text(_d.lower()),
    tags=[str(i)]) for i, _d in enumerate(train_set['
    final_request_text'])]

new_doc2vec_300_model = Doc2Vec(vector_size=300, min_count=2, epochs
    = 50)

new_doc2vec_300_model.build_vocab(new_tagged_data)
new_doc2vec_300_model.train(new_tagged_data, epochs=
    new_doc2vec_300_model.epochs, total_examples=new_doc2vec_300_model
    .corpus_count)

new_doc2vec_300_v = np.row_stack([new_doc2vec_300_model.infer_vector(
    process_text(train_set['final_request_text'][x])) for x in range(
    len(train_set))])
new_doc2vec_300_df = pd.DataFrame(new_doc2vec_300_v)
new_doc2vec_300_labels = train_set['requester_received_pizza']

new_skfold_doc2vec_300 = StratifiedKFold(n_splits = 10, shuffle=True,
     random_state=RANDOM_STATE)
new_doc2vec_300_model_lr = LogisticRegression(class_weight = {0:1,
    1:5}, random_state=RANDOM_STATE)
new_doc2vec_300_model_nbg = GaussianNB()
```

```
new_doc2vec_300_model_rf = RandomForestClassifier(n_estimators=590,
    min_samples_leaf=2, min_samples_split=2, max_features='sqrt',
    max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_doc2vec_300_model_lr_accuracy_score = []
new_doc2vec_300_model_nbg_accuracy_score = []
new_doc2vec_300_model_rf_accuracy_score = []

for train_index, test_index in new_skfold_doc2vec_300.split(
    new_doc2vec_300_df, new_doc2vec_300_labels):
     X_train, X_test = new_doc2vec_300_df.iloc[train_index,:],
         new_doc2vec_300_df.iloc[test_index,:]
     y_train, y_test = new_doc2vec_300_labels[train_index],
         new_doc2vec_300_labels[test_index]

     new_doc2vec_300_model_lr.fit(X_train, y_train)
     new_doc2vec_300_model_lr_accuracy_score.append(accuracy_score(
         y_test, new_doc2vec_300_model_lr.predict(X_test)))

     new_doc2vec_300_model_nbg.fit(X_train, y_train)
     new_doc2vec_300_model_nbg_accuracy_score.append(accuracy_score(
         y_test, new_doc2vec_300_model_nbg.predict(X_test)))

     new_doc2vec_300_model_rf.fit(X_train, y_train)
     new_doc2vec_300_model_rf_accuracy_score.append(accuracy_score(
         y_test, new_doc2vec_300_model_rf.predict(X_test)))


new_final_doc2vec_300_rf_df = final_train_set.copy()
new_doc2vec_300_rf_est_prob = new_doc2vec_300_model_rf.predict_proba(
    new_doc2vec_300_df)[:,1]
new_final_doc2vec_300_rf_df.insert(5,'prob_from_text',
    new_doc2vec_300_rf_est_prob)


new_final_doc2vec_300_rf_df = new_final_doc2vec_300_rf_df.sample(frac
    =1).reset_index(drop=True)
new_final_doc2vec_300_rf_labels = new_final_doc2vec_300_rf_df.iloc
    [:,-1]
```

```
new_final_doc2vec_300_rf_df = new_final_doc2vec_300_rf_df.iloc[:,:-1]
new_final_doc2vec_300_rf_labels


new_skfold_final_doc2vec_300 = StratifiedKFold(n_splits = 10, shuffle
    =True, random_state=RANDOM_STATE)
new_final_doc2vec_300_model_lr = LogisticRegression(class_weight =
    {0:1, 1:5}, random_state=RANDOM_STATE)
new_final_doc2vec_300_model_nbg = GaussianNB()
new_final_doc2vec_300_model_rf = RandomForestClassifier(n_estimators
    =590, min_samples_leaf=2, min_samples_split=2, max_features='sqrt'
    ,max_depth=50, bootstrap=False, random_state=RANDOM_STATE)

new_final_doc2vec_300_model_lr_accuracy_score = []
new_final_doc2vec_300_model_nbg_accuracy_score = []
new_final_doc2vec_300_model_rf_accuracy_score = []

new_final_doc2vec_300_model_lr_precision_score = []
new_final_doc2vec_300_model_nbg_precision_score = []
new_final_doc2vec_300_model_rf_precision_score = []

new_final_doc2vec_300_model_lr_recall_score = []
new_final_doc2vec_300_model_nbg_recall_score = []
new_final_doc2vec_300_model_rf_recall_score = []

new_final_doc2vec_300_model_lr_f1_score = []
new_final_doc2vec_300_model_nbg_f1_score = []
new_final_doc2vec_300_model_rf_f1_score = []


for train_index, test_index in new_skfold_final_doc2vec_300.split(
    new_final_doc2vec_300_rf_df, new_final_doc2vec_300_rf_labels):
    X_train, X_test = new_final_doc2vec_300_rf_df.iloc[train_index
        ,:], new_final_doc2vec_300_rf_df.iloc[test_index,:]
    y_train, y_test = new_final_doc2vec_300_rf_labels[train_index],
        new_final_doc2vec_300_rf_labels[test_index]

    new_final_doc2vec_300_model_lr.fit(X_train,y_train)
```

```
    new_final_doc2vec_300_model_lr_accuracy_score.append(
        accuracy_score(y_test, new_final_doc2vec_300_model_lr.predict(
        X_test)))
    new_final_doc2vec_300_model_lr_precision_score.append(
        precision_score(y_test, new_final_doc2vec_300_model_lr.predict(
        X_test)))
    new_final_doc2vec_300_model_lr_recall_score.append(recall_score(
        y_test, new_final_doc2vec_300_model_lr.predict(X_test)))
    new_final_doc2vec_300_model_lr_f1_score.append(f1_score(y_test,
        new_final_doc2vec_300_model_lr.predict(X_test)))

    new_final_doc2vec_300_model_nbg.fit(X_train, y_train)
    new_final_doc2vec_300_model_nbg_accuracy_score.append(
        accuracy_score(y_test, new_final_doc2vec_300_model_nbg.predict(
        X_test)))
    new_final_doc2vec_300_model_nbg_precision_score.append(
        precision_score(y_test, new_final_doc2vec_300_model_nbg.predict
        (X_test)))
    new_final_doc2vec_300_model_nbg_recall_score.append(recall_score(
        y_test, new_final_doc2vec_300_model_nbg.predict(X_test)))
    new_final_doc2vec_300_model_nbg_f1_score.append(f1_score(y_test,
        new_final_doc2vec_300_model_nbg.predict(X_test)))

    new_final_doc2vec_300_model_rf.fit(X_train, y_train)
    new_final_doc2vec_300_model_rf_accuracy_score.append(
        accuracy_score(y_test, new_final_doc2vec_300_model_rf.predict(
        X_test)))
    new_final_doc2vec_300_model_rf_precision_score.append(
        precision_score(y_test, new_final_doc2vec_300_model_rf.predict(
        X_test)))
    new_final_doc2vec_300_model_rf_recall_score.append(recall_score(
        y_test, new_final_doc2vec_300_model_rf.predict(X_test)))
    new_final_doc2vec_300_model_rf_f1_score.append(f1_score(y_test,
        new_final_doc2vec_300_model_rf.predict(X_test)))


# using shapley values to understand importance of each feature
doc2vec_300_explainer = shap.TreeExplainer(
    new_final_doc2vec_300_model_rf)
```

```
doc2vec_300_shap_values = doc2vec_300_explainer.shap_values(X_test)
shap.summary_plot(doc2vec_300_shap_values, X_test, plot_type="bar")
plt.savefig('eps/FeatureImportanceShapley_doc2vec_300.eps',dpi=1200,
    bbox_inches='tight')


sorted_idx = new_final_doc2vec_300_model_rf.feature_importances_.
    argsort()
plt.barh(new_final_doc2vec_300_rf_df.columns[sorted_idx],
    new_final_doc2vec_300_model_rf.feature_importances_[sorted_idx])


test_doc2vec_300_v = np.row_stack([new_doc2vec_300_model.infer_vector
    (process_text(test_set['final_request_text'][x])) for x in range(
    len(test_set))])
test_doc2vec_300_df = pd.DataFrame(test_doc2vec_300_v)


test_final_doc2vec_300_rf_df = final_test_set.copy()
test_tfidf_doc2vec_300_est_prob = new_doc2vec_300_model_rf.
    predict_proba(test_doc2vec_300_df)[:,1]
test_final_doc2vec_300_rf_df.insert(5,'prob_from_text',
    test_tfidf_doc2vec_300_est_prob)
test_final_doc2vec_300_rf_df


test_final_doc2vec_300_rf_df = test_final_doc2vec_300_rf_df.sample(
    frac=1).reset_index(drop=True)
test_final_doc2vec_300_rf_labels = test_final_doc2vec_300_rf_df.iloc
    [:,-1]
test_final_doc2vec_300_rf_df = test_final_doc2vec_300_rf_df.iloc
    [:,:-1]
test_final_doc2vec_300_rf_labels


# testing performance

final_accuracy = pd.DataFrame({'Count␣Model':[
```

```
            100*accuracy_score(
                test_final_count_rf_labels,
                new_final_count_rf_model_lr.predict(
                test_final_count_rf_df)),
            100*accuracy_score(
                test_final_count_rf_labels,
                new_final_count_rf_model_nbg.predict(
                test_final_count_rf_df)),
            100*accuracy_score(
                test_final_count_rf_labels,
                new_final_count_rf_model_rf.predict(
                test_final_count_rf_df))
        ],
        'TF-IDF':[
            100*accuracy_score(
                test_final_tfidf_rf_labels,
                new_final_tfidf_rf_model_lr.predict(
                test_final_tfidf_rf_df)),
            100*accuracy_score(
                test_final_tfidf_rf_labels,
                new_final_tfidf_rf_model_nbg.predict(
                test_final_tfidf_rf_df)),
            100*accuracy_score(
                test_final_tfidf_rf_labels,
                new_final_tfidf_rf_model_rf.predict(
                test_final_tfidf_rf_df))
        ],
        'Min-Max␣Doc2Vec':[
            100*accuracy_score(
                test_final_minmax_word2vec_rf_labels,
                 new_final_minmax_word2vec_model_lr.
                predict(
                test_final_minmax_word2vec_rf_df)),
            100*accuracy_score(
                test_final_minmax_word2vec_rf_labels,
                 new_final_minmax_word2vec_model_nbg.
                predict(
                test_final_minmax_word2vec_rf_df)),
```

```
            100*accuracy_score(
                test_final_minmax_word2vec_rf_labels,
                 new_final_minmax_word2vec_model_rf.
                predict(
                test_final_minmax_word2vec_rf_df))
        ],
        'Doc2Vec␣100':[
            100*accuracy_score(
                test_final_doc2vec_100_rf_labels,
                new_final_doc2vec_100_model_lr.
                predict(test_final_doc2vec_100_rf_df)
                ),
            100*accuracy_score(
                test_final_doc2vec_100_rf_labels,
                new_final_doc2vec_100_model_nbg.
                predict(test_final_doc2vec_100_rf_df)
                ),
            100*accuracy_score(
                test_final_doc2vec_100_rf_labels,
                new_final_doc2vec_100_model_rf.
                predict(test_final_doc2vec_100_rf_df)
                )
        ],
        'Doc2Vec␣300':[
            100*accuracy_score(
                test_final_doc2vec_300_rf_labels,
                new_final_doc2vec_300_model_lr.
                predict(test_final_doc2vec_300_rf_df)
                ),
            100*accuracy_score(
                test_final_doc2vec_300_rf_labels,
                new_final_doc2vec_300_model_nbg.
                predict(test_final_doc2vec_300_rf_df)
                ),
            100*accuracy_score(
                test_final_doc2vec_300_rf_labels,
                new_final_doc2vec_300_model_rf.
                predict(test_final_doc2vec_300_rf_df)
                )
```

```
                                ]},
                               index = ['LogReg','Gaussian␣NB','RF'])
final_precision = pd.DataFrame({'Count␣Model':[
                                    100*precision_score(
                                        test_final_count_rf_labels,
                                        new_final_count_rf_model_lr.predict(
                                        test_final_count_rf_df)),
                                    100*precision_score(
                                        test_final_count_rf_labels,
                                        new_final_count_rf_model_nbg.predict(
                                        test_final_count_rf_df)),
                                    100*precision_score(
                                        test_final_count_rf_labels,
                                        new_final_count_rf_model_rf.predict(
                                        test_final_count_rf_df))
                                ],
                                'TF-IDF':[
                                    100*precision_score(
                                        test_final_tfidf_rf_labels,
                                        new_final_tfidf_rf_model_lr.predict(
                                        test_final_tfidf_rf_df)),
                                    100*precision_score(
                                        test_final_tfidf_rf_labels,
                                        new_final_tfidf_rf_model_nbg.predict(
                                        test_final_tfidf_rf_df)),
                                    100*precision_score(
                                        test_final_tfidf_rf_labels,
                                        new_final_tfidf_rf_model_rf.predict(
                                        test_final_tfidf_rf_df))
                                ],
                                'Min-Max␣Doc2Vec':[
                                    100*precision_score(
                                        test_final_minmax_word2vec_rf_labels,
                                         new_final_minmax_word2vec_model_lr.
                                        predict(
                                        test_final_minmax_word2vec_rf_df)),
                                    100*precision_score(
                                        test_final_minmax_word2vec_rf_labels,
                                         new_final_minmax_word2vec_model_nbg.
```

```
            predict(
            test_final_minmax_word2vec_rf_df)),
        100*precision_score(
            test_final_minmax_word2vec_rf_labels,
             new_final_minmax_word2vec_model_rf.
            predict(
            test_final_minmax_word2vec_rf_df))
    ],
    'Doc2Vec 100':[
        100*precision_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_lr.
            predict(test_final_doc2vec_100_rf_df)
            ),
        100*precision_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_nbg.
            predict(test_final_doc2vec_100_rf_df)
            ),
        100*precision_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_rf.
            predict(test_final_doc2vec_100_rf_df)
            )
    ],
    'Doc2Vec 300':[
        100*precision_score(
            test_final_doc2vec_300_rf_labels,
            new_final_doc2vec_300_model_lr.
            predict(test_final_doc2vec_300_rf_df)
            ),
        100*precision_score(
            test_final_doc2vec_300_rf_labels,
            new_final_doc2vec_300_model_nbg.
            predict(test_final_doc2vec_300_rf_df)
            ),
        100*precision_score(
            test_final_doc2vec_300_rf_labels,
            new_final_doc2vec_300_model_rf.
```

```
                                predict(test_final_doc2vec_300_rf_df)
                                )
                        ]},
                        index = ['LogReg','Gaussian␣NB','RF'])

final_recall = pd.DataFrame({'Count␣Model':[
                            100*recall_score(
                                test_final_count_rf_labels,
                                new_final_count_rf_model_lr.predict(
                                test_final_count_rf_df)),
                            100*recall_score(
                                test_final_count_rf_labels,
                                new_final_count_rf_model_nbg.predict(
                                test_final_count_rf_df)),
                            100*recall_score(
                                test_final_count_rf_labels,
                                new_final_count_rf_model_rf.predict(
                                test_final_count_rf_df))
                        ],
                        'TF-IDF':[
                            100*recall_score(
                                test_final_tfidf_rf_labels,
                                new_final_tfidf_rf_model_lr.predict(
                                test_final_tfidf_rf_df)),
                            100*recall_score(
                                test_final_tfidf_rf_labels,
                                new_final_tfidf_rf_model_nbg.predict(
                                test_final_tfidf_rf_df)),
                            100*recall_score(
                                test_final_tfidf_rf_labels,
                                new_final_tfidf_rf_model_rf.predict(
                                test_final_tfidf_rf_df))
                        ],
                        'Min-Max␣Doc2Vec':[
                            100*recall_score(
                                test_final_minmax_word2vec_rf_labels,
                                 new_final_minmax_word2vec_model_lr.
                                predict(
                                test_final_minmax_word2vec_rf_df)),
```

```
            100*recall_score(
                test_final_minmax_word2vec_rf_labels,
                 new_final_minmax_word2vec_model_nbg.
                predict(
                test_final_minmax_word2vec_rf_df)),
            100*recall_score(
                test_final_minmax_word2vec_rf_labels,
                 new_final_minmax_word2vec_model_rf.
                predict(
                test_final_minmax_word2vec_rf_df))
        ],
        'Doc2Vec␣100':[
            100*recall_score(
                test_final_doc2vec_100_rf_labels,
                new_final_doc2vec_100_model_lr.
                predict(test_final_doc2vec_100_rf_df)
                ),
            100*recall_score(
                test_final_doc2vec_100_rf_labels,
                new_final_doc2vec_100_model_nbg.
                predict(test_final_doc2vec_100_rf_df)
                ),
            100*recall_score(
                test_final_doc2vec_100_rf_labels,
                new_final_doc2vec_100_model_rf.
                predict(test_final_doc2vec_100_rf_df)
                )
        ],
        'Doc2Vec␣300':[
            100*recall_score(
                test_final_doc2vec_300_rf_labels,
                new_final_doc2vec_300_model_lr.
                predict(test_final_doc2vec_300_rf_df)
                ),
            100*recall_score(
                test_final_doc2vec_300_rf_labels,
                new_final_doc2vec_300_model_nbg.
                predict(test_final_doc2vec_300_rf_df)
                ),
```

```
                    100*recall_score(
                        test_final_doc2vec_300_rf_labels,
                        new_final_doc2vec_300_model_rf.
                        predict(test_final_doc2vec_300_rf_df)
                        )
                ]},
                index = ['LogReg','Gaussian␣NB','RF'])
final_f1_score = pd.DataFrame({'Count␣Model':[
                    100*f1_score(test_final_count_rf_labels,
                         new_final_count_rf_model_lr.predict(
                        test_final_count_rf_df)),
                    100*f1_score(test_final_count_rf_labels,
                         new_final_count_rf_model_nbg.predict
                        (test_final_count_rf_df)),
                    100*f1_score(test_final_count_rf_labels,
                         new_final_count_rf_model_rf.predict(
                        test_final_count_rf_df))
                ],
                'TF-IDF':[
                    100*f1_score(test_final_tfidf_rf_labels,
                         new_final_tfidf_rf_model_lr.predict(
                        test_final_tfidf_rf_df)),
                    100*f1_score(test_final_tfidf_rf_labels,
                         new_final_tfidf_rf_model_nbg.predict
                        (test_final_tfidf_rf_df)),
                    100*f1_score(test_final_tfidf_rf_labels,
                         new_final_tfidf_rf_model_rf.predict(
                        test_final_tfidf_rf_df))
                ],
                'Min-Max␣Doc2Vec':[
                    100*f1_score(
                        test_final_minmax_word2vec_rf_labels,
                         new_final_minmax_word2vec_model_lr.
                        predict(
                        test_final_minmax_word2vec_rf_df)),
                    100*f1_score(
                        test_final_minmax_word2vec_rf_labels,
                         new_final_minmax_word2vec_model_nbg.
                        predict(
```

```
            test_final_minmax_word2vec_rf_df)),
        100*f1_score(
            test_final_minmax_word2vec_rf_labels,
             new_final_minmax_word2vec_model_rf.
            predict(
            test_final_minmax_word2vec_rf_df))
    ],
    'Doc2Vec␣100':[
        100*f1_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_lr.
            predict(test_final_doc2vec_100_rf_df)
            ),
        100*f1_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_nbg.
            predict(test_final_doc2vec_100_rf_df)
            ),
        100*f1_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_rf.
            predict(test_final_doc2vec_100_rf_df)
            )
    ],
    'Doc2Vec␣300':[
        100*f1_score(
            test_final_doc2vec_300_rf_labels,
            new_final_doc2vec_300_model_lr.
            predict(test_final_doc2vec_300_rf_df)
            ),
        100*f1_score(
            test_final_doc2vec_300_rf_labels,
            new_final_doc2vec_300_model_nbg.
            predict(test_final_doc2vec_300_rf_df)
            ),
        100*f1_score(
            test_final_doc2vec_300_rf_labels,
            new_final_doc2vec_300_model_rf.
            predict(test_final_doc2vec_300_rf_df)
```

```
                                    )
                            ]},
                           index = ['LogReg','Gaussian␣NB','RF'])

final_auc_roc = pd.DataFrame({'Count␣Model':[
                                    100*roc_auc_score(
                                        test_final_count_rf_labels,
                                        new_final_count_rf_model_lr.
                                        predict_proba(test_final_count_rf_df)
                                        [:, 1]),
                                    100*roc_auc_score(
                                        test_final_count_rf_labels,
                                        new_final_count_rf_model_nbg.
                                        predict_proba(test_final_count_rf_df)
                                        [:, 1]),
                                    100*roc_auc_score(
                                        test_final_count_rf_labels,
                                        new_final_count_rf_model_rf.
                                        predict_proba(test_final_count_rf_df)
                                        [:, 1])
                                ],
                                'TF-IDF':[
                                    100*roc_auc_score(
                                        test_final_tfidf_rf_labels,
                                        new_final_tfidf_rf_model_lr.
                                        predict_proba(test_final_tfidf_rf_df)
                                        [:, 1]),
                                    100*roc_auc_score(
                                        test_final_tfidf_rf_labels,
                                        new_final_tfidf_rf_model_nbg.
                                        predict_proba(test_final_tfidf_rf_df)
                                        [:, 1]),
                                    100*roc_auc_score(
                                        test_final_tfidf_rf_labels,
                                        new_final_tfidf_rf_model_rf.
                                        predict_proba(test_final_tfidf_rf_df)
                                        [:, 1])
                                ],
                                'Min-Max␣Doc2Vec':[
```

```
        100*roc_auc_score(
            test_final_minmax_word2vec_rf_labels,
             new_final_minmax_word2vec_model_lr.
            predict_proba(
            test_final_minmax_word2vec_rf_df)[:,
            1]),
        100*roc_auc_score(
            test_final_minmax_word2vec_rf_labels,
             new_final_minmax_word2vec_model_nbg.
            predict_proba(
            test_final_minmax_word2vec_rf_df)[:,
            1]),
        100*roc_auc_score(
            test_final_minmax_word2vec_rf_labels,
             new_final_minmax_word2vec_model_rf.
            predict_proba(
            test_final_minmax_word2vec_rf_df)[:,
            1])
    ],
    'Doc2Vec␣100':[
        100*roc_auc_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_lr.
            predict_proba(
            test_final_doc2vec_100_rf_df)[:, 1]),
        100*roc_auc_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_nbg.
            predict_proba(
            test_final_doc2vec_100_rf_df)[:, 1]),
        100*roc_auc_score(
            test_final_doc2vec_100_rf_labels,
            new_final_doc2vec_100_model_rf.
            predict_proba(
            test_final_doc2vec_100_rf_df)[:, 1])
    ],
    'Doc2Vec␣300':[
        100*roc_auc_score(
            test_final_doc2vec_300_rf_labels,
```

```
                                    new_final_doc2vec_300_model_lr.
                                    predict_proba(
                                    test_final_doc2vec_300_rf_df)[:, 1]),
                                100*roc_auc_score(
                                    test_final_doc2vec_300_rf_labels,
                                    new_final_doc2vec_300_model_nbg.
                                    predict_proba(
                                    test_final_doc2vec_300_rf_df)[:, 1]),
                                100*roc_auc_score(
                                    test_final_doc2vec_300_rf_labels,
                                    new_final_doc2vec_300_model_rf.
                                    predict_proba(
                                    test_final_doc2vec_300_rf_df)[:, 1])
                            ]},
                           index = ['LogReg','Gaussian␣NB','RF'])



font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 12}

plt.rc('font', **font)
fig, ax = plt.subplots(2,2, figsize=(15,14))
ax[0][0].plot(final_accuracy.index,final_accuracy.loc[final_accuracy.
    index,:], marker = 'o')
ax[0][0].legend(final_accuracy.columns)
ax[0][0].set_ylim(0,100)
ax[0][0].set_title('Accuracy')
ax[0][1].plot(final_precision.index,final_precision.loc[
    final_precision.index,:], marker = 'o')
ax[0][1].legend(final_precision.columns)
ax[0][1].set_ylim(0,100)
ax[0][1].set_title('Precision')
ax[1][0].plot(final_recall.index,final_recall.loc[final_recall.index
    ,:], marker = 'o')
ax[1][0].legend(final_recall.columns)
ax[1][0].set_ylim(0,100)
ax[1][0].set_title('Recall')
```

```
ax[1][1].plot(final_f1_score.index,final_f1_score.loc[final_f1_score.
    index,:], marker = 'o')
ax[1][1].legend(final_f1_score.columns)
ax[1][1].set_ylim(0,100)
ax[1][1].set_title('F-Score')
fig.show()
fig.savefig('eps/Final_Model_Performance.eps',dpi=1200, bbox_inches =
     'tight')



plt.figure(figsize = (12,10))
plt.plot(final_auc_roc.index,final_auc_roc.loc[final_auc_roc.index
    ,:], marker = 'o')
plt.legend(final_auc_roc.columns)
plt.ylim(0,100)
plt.title('AUC␣ROC␣Score')
plt.savefig('eps/Auc_Roc_Score_metric.eps', dpi = 1200, bbox_inches =
     'tight')
plt.show()


# Graph for Text Models

text_accuracy_score = {'Count␣Model':[accuracy_score(final_test_set['
    requester_received_pizza'],new_count_model_lr.predict(
    test_count_df))*100,accuracy_score(final_test_set['
    requester_received_pizza'],new_count_model_nbg.predict(
    test_count_df.toarray()))*100,accuracy_score(final_test_set['
    requester_received_pizza'],new_count_model_rf.predict(
    test_count_df))*100],
                       'TF-IDF':[accuracy_score(final_test_set['
                           requester_received_pizza'],
                           new_tfidf_model_lr.predict(test_tfidf_df))
                           *100,accuracy_score(final_test_set['
                           requester_received_pizza'],
                           new_tfidf_model_nbg.predict(test_tfidf_df.
                           toarray()))*100,accuracy_score(
                           final_test_set['requester_received_pizza'],
```

```
                                new_tfidf_model_rf.predict(test_tfidf_df))
                                *100],
                            'MinMax␣Word2Vec':[accuracy_score(
                                final_test_set['requester_received_pizza'],
                                new_minmax_word2vec_model_lr.predict(
                                test_minmax_word2vec_df))*100,
                                accuracy_score(final_test_set['
                                requester_received_pizza'],
                                new_minmax_word2vec_model_rf.predict(
                                test_minmax_word2vec_df))*100,
                                accuracy_score(final_test_set['
                                requester_received_pizza'],
                                new_minmax_word2vec_model_rf.predict(
                                test_minmax_word2vec_df))*100],
                            'Doc2Vec␣100':[accuracy_score(final_test_set['
                                requester_received_pizza'],
                                new_doc2vec_100_model_lr.predict(
                                test_doc2vec_100_df))*100,accuracy_score(
                                final_test_set['requester_received_pizza'],
                                new_doc2vec_100_model_nbg.predict(
                                test_doc2vec_100_df))*100,accuracy_score(
                                final_test_set['requester_received_pizza'],
                                new_doc2vec_100_model_rf.predict(
                                test_doc2vec_100_df))*100],
                            'Doc2Vec␣300':[accuracy_score(final_test_set['
                                requester_received_pizza'],
                                new_doc2vec_300_model_lr.predict(
                                test_doc2vec_300_df))*100,accuracy_score(
                                final_test_set['requester_received_pizza'],
                                new_doc2vec_300_model_nbg.predict(
                                test_doc2vec_300_df))*100,accuracy_score(
                                final_test_set['requester_received_pizza'],
                                new_doc2vec_300_model_rf.predict(
                                test_doc2vec_300_df))*100]}
plot_data = pd.DataFrame(text_accuracy_score, index=['Logistic␣
    Regression','Gaussian␣Naive␣Bayes','Random␣Forest'])

fig = plt.figure(figsize=(10,5))
```

```
plt.plot(plot_data.index,plot_data.loc[plot_data.index,:], marker = '
    o')
plt.ylim(40,100)
plt.title('Comparison␣of␣Accuracy␣Score␣using␣different␣Text␣
    Embedding␣Methods')
plt.xlabel('Models/Algorithms␣Used')
plt.ylabel('Accuracy␣Score')
plt.legend(plot_data.columns)
plt.savefig('eps/AccScoreTextModel.eps', dpi=1200,bbox_inches='tight'
    )
plt.show()
```