Exploration of Useful Comments in Stack Overflow

1. Introduction

Apart from the questions and the answers, comments which the community members discuss about particular problems also play an important role in Stack Overflow. These code comments can improve the software maintainability by helping the developers to understand the code segments posted and this has become a standard practice used in the industry. But some code bases do not consist of the adequate amount of codes or any other insightful comments which do not relate to the problem directly. They are not important in the context of searching for a solution. Most developers tend to post comments in Stack Overflow without properly following the Standard Commenting Guidelines of the Q&A platform. Useful Comments can be defined as the comments in Stack Overflow, which adheres to Stack Overflow's commenting guidelines and which belongs to both questions and answers. Therefore, it is important to explore Useful Comments for the purpose of improving the User Experience in Stack Overflow.

2. Background and motivation

When dealing with the user experience offered by Stack Overflow, developer-discussions pertain to a major role. Code comments are considered as the key factor in which the developers communicate in Stack Overflow. Such comments which deal with scopes for further improvements of the source code, assist developers to analyze the code further, for the purpose of understanding code reuse [1]. There is a tendency for such comments to remain unattended due to various reasons which let fall the user experience of Stack Overflow.

3. Problem in Brief

Importance of developer discussions i.e. comments in Stack Overflow revolve around many important aspects. Code comments that provide insight into the scopes or issues and quality for further improvement of the source code assist developers to analyze the code further in understanding reuse of code and change tasks. Comments in Stack Overflow are important for the recommendation of source code comment [1]. Improving software maintainability is enabled through comments which supports the developers in understanding the code. In Stack Overflow

because of the large user base, questions posted frequently receive high quality answers and also code descriptions that can be used in software projects for automatic comment generation [4]. By mining comments in Stack Overflow, translation of emotion mining into actionable sights is made possible [3].

In Stack Overflow, there is a standardized method for the developers to add comments. Such comments are understood by several categories of comments which include;

- Request clarification from the author [6]
- Constructive criticism which guides the author in improving the post [6]
- Relevant but transient information, such as a link to a related question [6]

Unfortunately, it is apparent that most of the developers tend to comment on posts with comments which belong to categories other than the above mentioned categories of code comments. Such comments can be known as trivial comments [1]. Hence, there is a tendency where useful comments get ignored by the author, in Stack Overflow. According to previous studies, 27.5% of the comments which warrant an update were ignored [2]. Furthermore, from the above standard categories of comments in Stack Overflow, previous studies were conducted upon comments which guide for an answer update and only on comments that are associated with answers [2].

Therefore there is a need that Stack Overflow designers should improve the present commenting system since users post comments in unrecommended manners by not following the standard commenting guidelines presented by Stack Overflow. A systematic study has not ever been performed in the comments in Stack Overflow to understand better the usage of comments. There is a need for data-driven solutions to retrieve useful comments which are informative to either summarize or identify such comments in a manner which is automated. Automatic identification of such comment types using Machine Learning and Natural Language Processing is yet to be addressed [6].

4. Related work

Previous studies related to the comments in Stack Overflow revolves around recommending insightful comments for Source Code by utilizing Crowdsourced Knowledge [1], Generation of

Automatic Comments for a complete java method [1], [4], studies regarding comment-induced answer updates in Stack Overflow, analysis of commenting activities in terms of timing, content and individuals[6], Coding schema which comprises 9 comment categories for the provision of insights into commenting in Stack Overflow [7], Identification of inadequate comments in Wikipedia's talk page edits and classification of these into different categories[9], Classification of Stack Overflow Posts by means of Contextual Tagging[10], Automating the classification of posts into 7 question categories in Stack overflow [11], Online Toxic Comment classification by making use of Logistic Regression and 3 Artificial Neural Network Approaches[13], Empirical study of the obsolete knowledge on Stack Overflow [12], system that predicts who will answer a certain question in Stack Overflow [8], Exploration of the means of which comments affect answer updates [2], A Gold Standard for the purpose Emotion Annotation in Stack Overflow [3], Automatic approach for Comment Generation which mines comments from Q&A sites [4], Extraction of candidate method documentation from Stack Overflow discussions and creation of JavaDoc descriptions [5], Analysing any piece or section of text and detection and observation of different types of toxicity [14] and utilizing the One-Vs-One decomposition strategy addressing the problem of overlapping data in classification [16]. The table below consists of further information related to the aforementioned different approaches related to exploration of comments in Stack Overflow, as well as the related approaches in comment classification.

	Existing Solutions	Important Facts	Limitations
1.	Analysis of commenting activities in respect of timing, content and individuals who perform commenting focusing on the comments which were posted in answers in Stack Overflow [6].	Comment classification was done by a light-weight open coding process in which 2 authors were involved in deriving a draft list of comment types. The time in which users take to post comments once the answer is posted is taken into consideration. The role of a	Need of a methodical and organized study related to the comments in Stack Overflow to better understand how comments are being used (for example, whether users use comments by following Stack Overflow commenting guidelines). Moreover, improving the current

		commenter is categorized as, 1. Asker: the user who posted the question. 2. Answerer: the user who posted the answer. 3. Outsider: the user who belongs to neither of the two above mentioned roles. [6].	commenting system is required since users post comments in unrecommended manners. Furthermore future research may leverage approaches from Machine Learning and NLP communities to automatically identify such comment categories [6]
2	Coding schema which comprises 9 comment categories for the purpose of provision of insights into commenting in Stack Overflow [7].	Qualitative analysis was conducted between 2 coders for the purpose of comment classification. Krippendorff's alpha was found to be 0.762 (related to the level of agreement between 2 coders) [7].	Further research on comment usage is needed [7]
3	Identification of inadequate comments in Wikipedia's talk page edits and classification of the same into different categories [9].	Accuracy Measures of Binary Relevance(BR) and Classifier Chain (CC) methods respectively are as follows [9]. ● Logistic Regression 98.145 (BR) 98.454(CC) ● KNN 98.386 (BR) 98.864(CC)	-

		• SVM 98.972 (BR) 98.991(CC) SVM provided the best results out of the utilized classification algorithms i.e. logistic regression, support vector machine (SVM), K nearest neighbour and decision tree [9]	
4	Classification of Stack Overflow Posts by utilizing Contextual Tagging mechanism[10]	Tagging posts based on the purpose and context Classifying questions was done previously using automatic categorization by topic modelling using LDA and MALLET, KNN Clustering, Manual Categorization. Derivation of 6 comment categories was performed utilizing the existing studies and the results which were obtained using topic modelling. SVM promised the highest accuracy of 78.5% in this approach [10].	Limited number of posts were examined during the study. Furthermore, Some of the statistical distributions were were not balanced and they were biased towards one certain topic [10]
5	Automation of the classification of posts into 7 categories of	Domain used was only Android. Manual Analysis of phrases was performed to	There is a possibility of Manual Categorization of the posts to be biased [11]

	questions in Stack overflow [11]	find patterns and training classification models based on Machine Learning Algorithms was done. Moreover, Accuracy Measure of Random forest model was 0.86 whereas the accuracy of SVM was 0.84 [11]	
6	Online Toxic Comment classification by making use of 3 Artificial Neural Network Approaches and Logistic Regression[13]	Logistic Regression assured the accuracy of 0.91 whereas convolutional neural network + long short-term memory approach promised the accuracy of 0.9820 [13]	-
7	Empirical study related to the obsolete knowledge on Stack Overflow [12]	Quantitative as well as a qualitative analysis related to the obsolete answers of Stack Overflow was conducted. The utilized heuristics based approach contained the accuracy of 75% [12]	Need of a Machine Learning based approach for the purpose of improving accuracy is considered as a limitation in the study. Moreover, need of more studies to acquire useful information from comments in Stack Overflow is required [12]
8	Prediction related to who will answer a specific question in Stack Overflow [8]	The related work of this approach included • Construction of	A hybrid approach which amalgamates both knowledge from the question and the asker to retrieve more

		co-occurrence graphs from tagging, voting, answering, commenting and users availability to predict a group of users who would be willing to collaborate to that question, Usage of Bayesian networks for the purpose of modelling a dependency between User Question-Answer to recommend a given question to appropriate answers, Mining communications among users (questioners and answerers) for the purpose of building a network of users and Feature-based classification for question routing. The feature based classification had a precision of 0.44 [8]	error-free candidate lists is needed [8]
9	Mining approach which suggests insightful comments in Stack Overflow [1]	Insightful comments are identified as comments which include scopes or quality deficiencies. This approach helps in source code comment recommendation Furthermore, the mining	Dataset which was considered for the empirical evaluation was limited. During the study, only 4 developers participated in the user study. It is stated that the approach might be inadequate

		approach assured the accuracy and precision of 80% [1]	in recommending comments for the code segment from proprietary or legacy projects. Furthermore the mining of insightful comments related to Stack Overflow was performed in the following domains only, which is Java, Android and C# and is considered as a limitation. The Comment classification is also bit biased[1]
10	Exploration of the means of which comments have an effect on answer updates [2]	Recognition of candidate comments and categorization of comments based on heuristics was performed [2].	Comments and answer updates which involve code segments were only being taken into consideration during the study. There is a tendency for the comments to be mislabeled if the code element in the comment is not correctly identified by the system, which leads to false positives [2]
11	A Gold Standard for Emotion Annotation in Stack Overflow [3]	Manual Annotation of Stack Overflow gold standard data set with emotion labels was performed. Identification of Emotions is based on clear guidelines of a conceptual	The gold standard on emotion polarity was built by utilizing manual annotation. There is a tendency for text items which contain interesting emotional

		framework which is based on theory. Moreover, Final gold labels were assigned through agreement of the majority of 3 coders[3].	content to be filtered out[3].
12	Automatic Comment Generation approach which mines comments from Q&A sites [4]	Code Description Mapping Extraction, Refinement of Description, Code Clone Detection, Code Clone Pruning, Selection of Comments were performed. For the Strongly agreed responses Java Domain consisted of accuracy of 79/150 whereas the Android Domain contained the accuracy of 34/75. The related work of this approach were Automatic comment generation for certain code structures such as exceptions, failed test cases, code changes and method parameters, Generation of Automatic Comments for an entire java method and Automatic comment generation for software concerns, Java classes, MPI	Failure at identifying comments that comprises an incorrect description of the code segment was stated as a limitation in this study. User Expectation with respect to the description of the surrounding code of the target code by using comments. Lengthy sentences were generated and the Comments were too trivial [4].

		methods and high level actions within methods [4].	
13	Extraction of candidate method documentation from discussions of Stack Overflow and creation of JavaDoc descriptions [5]	Precision of manual validation of eclipse was 79% whereas the precision of manual validation of Lucene was 87% [5]	Mining source code descriptions from developers' discussions need more improvement when regarding its usability [5]
14	Analysing any section of text and detecting distinct types of toxicity [14]	Wikipedia Comment Dataset by Jigsaw is made use of during the study. Accuracy comparison of the utilized approaches are as follows: - Tf-idf with 6 headed Machine Learning - 98.98% - CNN -91.2% - LSTM-92.7% - CNN with character level embeddings - 94% - LSTM with custom embeddings - 97.78% [14]	Grid search Algorithm can be utilized to obtain more accurate results [14].
15	By utilizing the One-Vs-One decomposition strategy addressing the problem of overlapping data in	The One-Vs-One(OVO) strategy results in higher performance when compared with Non-OVO approaches, disregarding the overlapping	Development of data cleaning methods for the purpose of reducing the difficulties in overlapping regions and sample weighting solutions

	classification[16]	level. It also benefits the multi-class classification. It also increases the separability of classes. Moreover, OVO highly benefits SVM, C4.5 and RIPPER classifiers as it provides robust results and superior performance with respect to all levels of overlapping [16].	for the purpose of reducing the influence of overlapping samples on the decision boundaries provided by the classifiers. Combining the aforementioned approaches with decomposition strategies are to be done further [16].

Table 4.1: Existing Solutions' detailed information and comparison of Useful Comments in Stack Overflow

5. Inputs to the Module

Sub Module	Inputs
Sampling	Comments in Stack Overflow
Qualitative Analysis and Review	Stack Overflow's Comments sample
Pre-processing	Noisy Data (Data after the qualitative analysis i.e. deriving the useful comments) which needs cleaning
Feature Extraction	Pre-processed useful comment text
Useful-Comment Categorizing	Extracted Features

Table 5.1 - Inputs to the Exploration of Useful Comments module

6. Outputs

Sub module	Output
Sampling	Comments sample of Stack Overflow
Qualitative Analysis and Review	Set of labeled Useful Comments
Pre-processing	Useful Comments data after pre-processing
Feature Extraction	Extracted Features of the useful comment text
Useful-Comment Categorizing	Useful Comment Categories

Table 6.1 - Outputs to the Exploration of Useful Comments module

7. Analysis and Design

Useful Comments can be defined as the comments in Stack Overflow, which adheres to Stack Overflow's commenting guidelines and which belongs to both questions and answers. Useful Comments include Request Clarification Comments, Constructive Criticism Comments and Relevant Information Comments. This module will be categorizing the comments in Stack Overflow to useful comment categories in which they fall into. The module comprises the following steps.

1. Sampling
2. Qualitative analysis and Review
3. Preprocessing
4. Feature Extraction
5. Training of the Useful Comment Classification Model

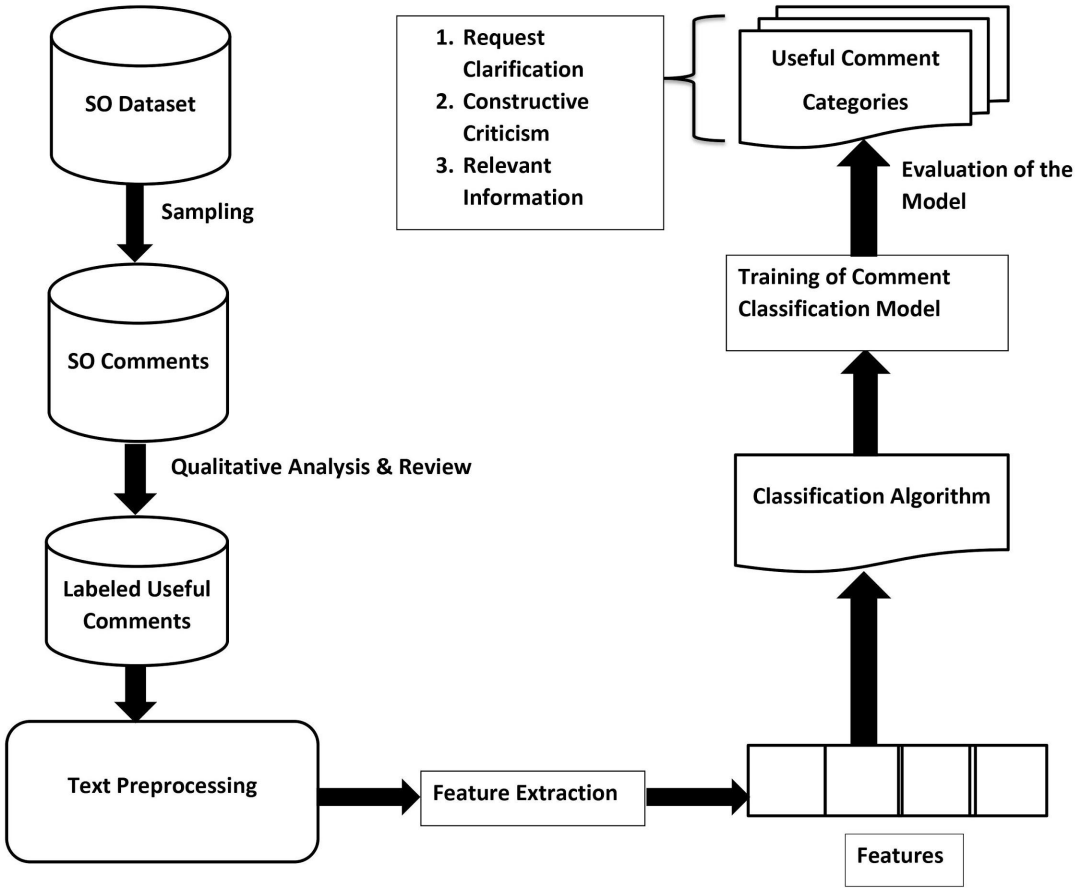


Figure 7.1 : High Level Architecture Diagram of Exploration of Useful Comments in Stack Overflow

1. Sampling

From the Data dump in Stack Overflow which is the Stack Exchange Data Explorer, Comments posted in Stack Overflow during the past 5 years i.e. comments posted after 1st of January 2015 and before 08th of November 2020 was taken into consideration during the study. 6164 comments were sampled using the simple random sampling technique which is a probability sampling technique that utilized randomization. This technique was used as there was no prior knowledge related to the comments data and therefore each element contained an equal chance of getting selected for the purpose of being a part of the sample [15].

2. Qualitative Analysis and Review

A total of 6164 comments were taken for the Qualitative Analysis.The qualitative analysis was done to label the comments according to specific definitions of the standard comment categories in Stack Overflow and to filter out the Useful Comments from the Comments which do not follow the Commenting guidelines of Stack Overflow. Deductive coding approach was performed since there is a direct focus on the study which is the comments which follow the Stack Overflow's commenting guidelines and the guidelines are mentioned in prior literature as well as the Stack Overflow's official website [17] itself. After completing the qualitative analysis on all 6164 comments initially 3587 useful comments were identified. Moreover single labeling is performed during the qualitative analysis which implies Multi Class Classification.

The following table discusses the measures in which qualitative analysis was conducted to the comments in Stack Overflow for labeling the identified useful comments in the standard comment categories.

Useful Comment Category i.e. Comment Labels	**Property Description**
Request Clarification - Requesting Clarification from the author [18]	- request / ask to provide more information for better understanding - Expression of lack of understanding - Can be expressed through confirmation of understanding E.g - " Let me see if I understood you correctly. You're saying that ..." - Can be identified with the keywords such as 'please clarify', 'please

	elaborate', 'how', 'what' etc.
Constructive Criticism - Guiding the author in improving the post [18]	- Offering valid and well-reasoned opinions - Includes both positive and negative comments - Stated in a friendly manner and pleasant manner rather than an oppositional one - Avoid dealing with personality issues of developers - Should be stated in accordance with recipient's work - Positive commenting through the usage of softer language. - Areas of improvement can be stated - Key points should be reasoned out objectively - Provision of recommendations for improvement - Assumptions should not be stated - May include formatting issues, indentation issues etc.
Relevant Information - Relevant but minor or transient	- A link to a related question

information to a post [18]	- A link to a related post - Informative links which redirects to other websites - Stating about Question updates (rarely found) - Stating about answer updates (rarely found) - Links to related posts were mostly identified through phrases such as 'possible duplicate of', 'duplicate of' etc.

Table 7.1 - Qualitative Analysis related to Useful Comments in Stack Overflow for labeling of useful comments

Quality of the labeled data can be ensured and measured in 3 different ways [19]. Namely,

I. Benchmarks[19]
II. Consensus[19]
III. Review[19]

Benchmarks measure the accuracy of the labeled data by taking a subset of the training data into consideration, while Consensus measures the inter annotator agreement between multiple annotators. Review is another measure to ensure the accuracy of labeling. The Review was chosen as the method of ensuring the quality of the labeled data, as it seemed to be more appropriate in this context. After completing the qualitative analysis it was necessary to review the labeling process of the comments in order to measure the consistency and accuracy of the labeled data [19]. Since the labeling and review was performed by a single annotator intra rater reliability percentage was calculated. As intra rater reliability is the degree of agreement among repeated administrations of a specific task performed by a single rater (same assessment is completed by the same rater on 2 or more occasions), it was considered as a measure to be relied upon in this context. Therefore, as a result of the review 3120 comments were properly labeled

with the intra rater reliability of 86.98% with regard to correct labeling of comments in both occasions. Therefore 467 comments were disregarded in the study further as they were identified as misclassified in the review, while resulting in 3120 useful comments to be used in the implementation of the classification model.

3. Preprocessing

The sampled and the identified useful comments might have noisy data. Thus, data cleaning is needed. Before the preprocessing stage, a feature engineering stage was performed which included feature creation and evaluation of the created features. The created features included length of comments, comment score, the punctuation percentage of the comments, average word count, the count of capital letters, the count of stop words and the sentiment score of comments. The sentiment score of comments were calculated using the NLTK's VADER which is a rule based model that is parsimonious. The feature evaluation resulted that each feature has the majority overlapping effect in 2 or more specified comment classes thus resulting in the features created through feature engineering to be disregarded in the further development of the model.

During the preprocessing stage the identified useful comments data was preprocessed using Natural Language Processing Techniques which include lowercasing the data, replacing URLs with a keyword (as URLs play an important role in the classification task of the Relevant Information comment category), removal of punctuation, stop words removal, tokenization of data, stemming and lemmatization, removal of numbers, removal of emojis and emoticons in comments, and handling of chat words identified. In this context, chat words are known as the abbreviations used in comments and handling of chat words include the replacement of the phrases that correspond to each chat word by reading a text document which includes a list of chat words. This chat words document was made by considering the glossary dictionary of stack exchange and some common chat words that came across during the qualitative analysis.

4. Feature Extraction

The preprocessed comment data were then utilized in the Feature Extraction Process. In this step comment text i.e. words of each comment were taken as the features since the numeric features gained through feature engineering were exposed to overlapping of data in several comment categories. The TF-IDF feature extraction technique was utilized for the text feature extraction process along with N-grams. For the N-grams features unigrams and bigrams were utilized as they promised an increase in accuracy score of the classifier. Furthermore, both tf-idf scores and n-grams added up to 275 features.

TF-IDF calculates the term frequency and inverse document frequency. IDF suppresses the effect of words which occurs in all 3 Comment Categories. Moreover, the TF-IDF vector looks the same as the word vector. This mechanism is used to find the meaning of sentences and it cancels out the incapableness of the bag of words feature extraction technique.

The TF Score of a particular word(w) and the IDF score can be computed as follows.

TF(w) = (Number of times term w appears in a particular document) / (Total number of terms in the document).

IDF(w) = log_e(Total number of documents / Number of documents with term w in the document)

TF answers how many times a particular word is used in the entire document, whereas IDF calculates the importance of a certain term in a list of documents.For the feature extraction purpose TF-IDF Vectorizer was used as it performs the task of count vectorizer which is followed by TF-IDf transformer.

For the purpose of boosting the accuracy, N-grams were used. The N-gram frequency method provided an inexpensive and highly effective method of classifying documents.

Therefore, as the combination of tf-idf scores and n-grams were used, it is most commonly known as N-gram level TF-IDF. This matrix represents tf-idf scores of N-grams.

5. Training of the Useful Comment Classification Model

The extracted features were fed into the Classification model for the training process after the train-test split. 80% of data was utilized for the training purpose and 20% was utilized for the testing purpose. After the train-test split 2496 comments were used for training and 624 comments were used for testing of the Classification Model. Support Vector Machine (SVM) Classification Algorithm was used to derive the comment category in which the Useful Comments belong to. Initially the SVM model without any parameters was used to train the model. For the initial model the accuracy scores, f1 scores, recall, precision, confusion matrix and Classification Report was gained. Afterwards hyperparameter tuning with 3-fold cross validation using Grid-SearchCV was done to identify the best parameter combination with each SVM Kernel and training with the best parameters for each SVM Kernel was performed. Since GridSearchCV utilizes k- folds cross validation for identifying the optimal combination of hyperparameters a separate validation data set was not used. This is because k-1 folds are used for training and the remaining fold is used for Testing in GridSearchCV. Identification of the best SVM Kernel that fits the problem context was done through accuracy measures and evaluation measures. The utilized Kernels are as follows.

I. Linear Kernel - Mostly Used when a large number of data is available and when the data is linearly separable. It is the simplest Kernel Function in SVM.
When x, xj are the data to be classified, the Linear Kernel function is as below.

$$F(x, xj) = sum(x.xj)$$

II. RBF(Radial Basis Function) Kernel - It is usually used in classifying non-linear data. Proper separation of data when there is no prior knowledge about the data is performed successfully. The formula of RBF is as follows.

```
K(x, y) = exp(-gamma ||x-y||^2)
```

Note that gamma varies between 0 and 1.

III. Polynomial Kernel - This kernel is a generalized representation of the Linear kernel. The formula is as follows.

```
K(X, Y) = (gamma <X, Y> + coef0)^degree
```

IV. Sigmoid Kernel - This is often known as hyperbolic tangent or Multilayer perceptron. This kernel is mostly preferred in Neural Networks.

```
K(X, Y) = tanh(gamma <X, Y> + coef0)
```

V. Precomputed Kernel - This kernel is used by passing the Gram Matrix.

After training, evaluation of each model with 5 different SVM Kernels (Linear, RBF, Polynomial, Sigmoid and Precomputed) was performed using the Holdout Method, Confusion Matrix and Classification Report.

8. Implementation

Supervised Machine Learning is a subcategory of Artificial Intelligence and Machine Learning. In Supervised Learning, Labeled data sets are used to train Classification Algorithms and to accurately predict outcomes. It helps in solving a variety of real world problems which are at scale. In the context of exploring comments which follow the Stack Overflow's Commenting guidelines i.e. Comments that are being defined as Useful Comments in this study, it is necessary to identify what each comment mean and it is necessary to manually label the comments data as there exists a predefined set of Useful Comment Categories as mentioned in the Stack Overflow's Commenting guidelines which is the priority of interest in this study.

Classification is identified as the solution for the predictive modeling problem of exploring Useful Comments in Stack Overflow. Since it falls into a subset of data driven approaches, it can be used to identify Useful Comments in Stack Overflow by categorizing such comments. Mainly text classification is used in this context as the comment text is categorized into the Standard Comment Categories in Stack Overflow. Multi Class Classification is used and thus each derived Useful Comment is given one specific category from the three Useful Comment Categories specified.

Support Vector Machine Algorithm explores the hyperplane in a specific N-dimensional space which classifies the data points in a distinct manner. Here, the number of features are denoted by N. SVM was utilized in classifying Useful Comments in Stack Overflow as it promised a higher accuracy [9] [10] in past research.

The implementation facts of this module is described in detail below.

8.1. Obtaining comments data through querying and Qualitative Analysis with Review

Comments in Stack Overflow data dump which were being posted during the past 5 years i.e. comments posted after 1st of January 2015 and before 08th of November 2020 were being taken into consideration in the study. Moreover, for the purpose of deriving the comments which have been posted in Stack Overflow's posts with importance, the comments which are being posted in the posts that contain tags and with the score which is greater than 0 are being taken into consideration. The comments data set was obtained by querying in Stack Exchange Data Explorer.

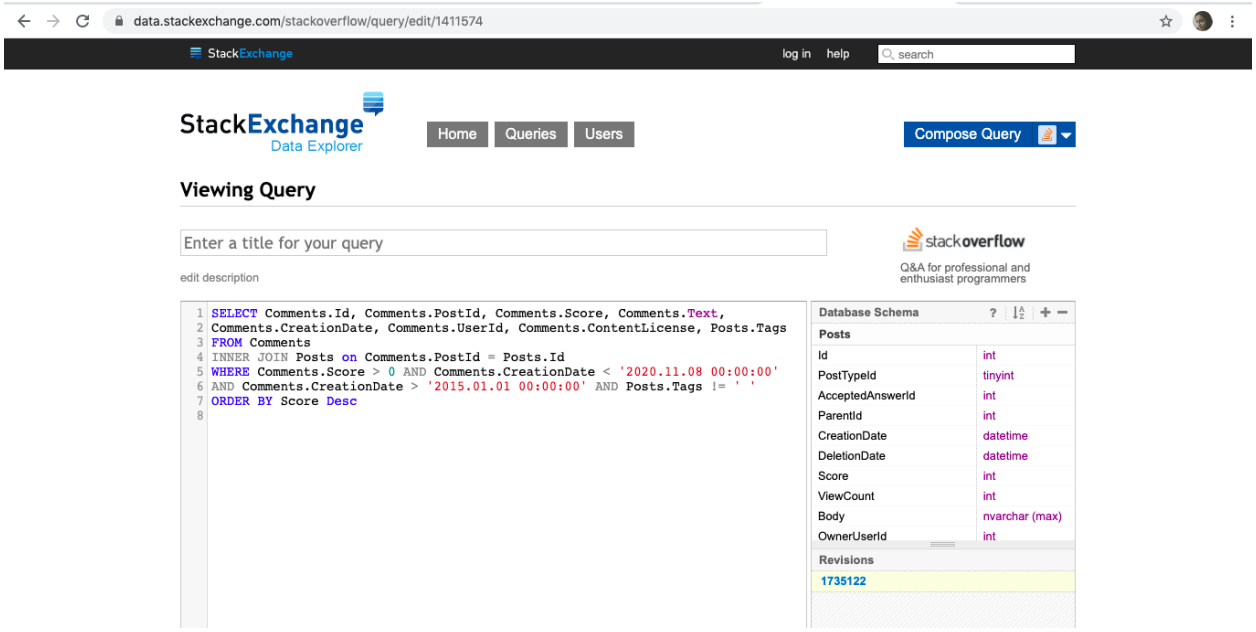
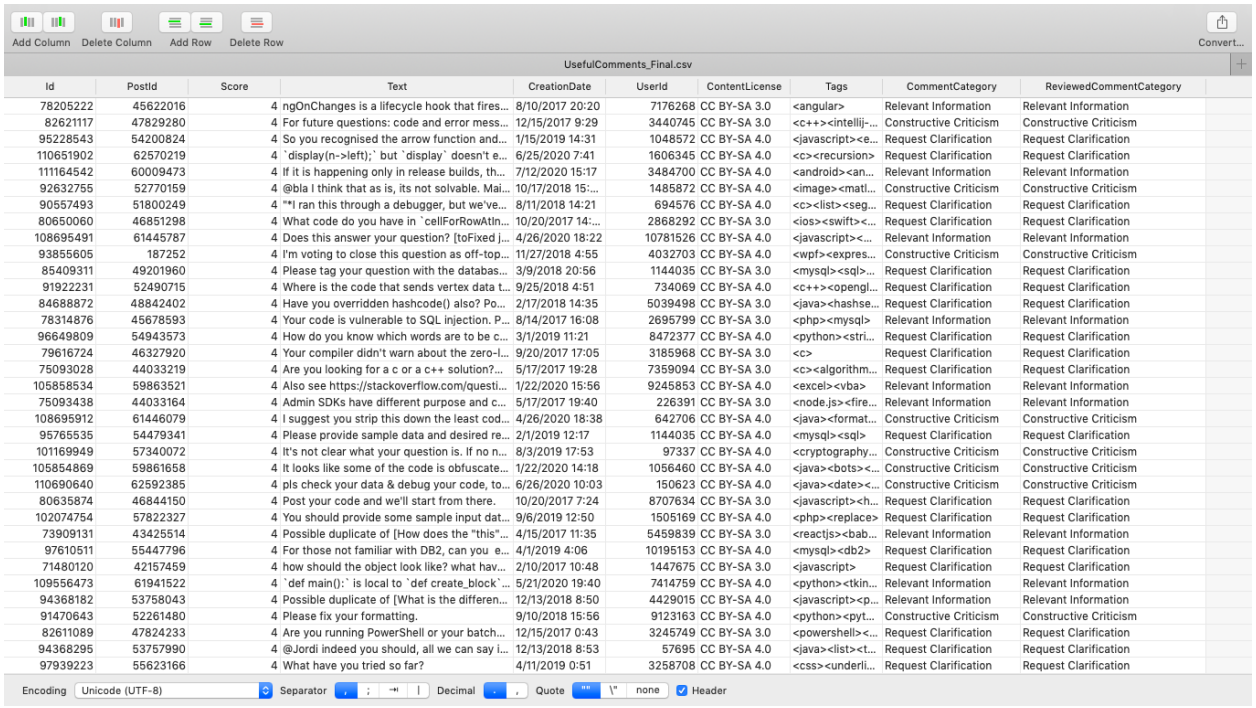


Figure 8.1 : Query utilized in obtaining comments from Stack Exchange Data Explorer

Since 6164 comments will be taken into consideration in this study, out of the 6164 comments which will be taken into consideration for the purpose of exploring the useful comments in Stack Overflow, qualitative analysis was performed on the entire comments set of 6164 (which were gained by simple random sampling) for the purpose of labeling the comment data. 3120 useful comments were derived after the qualitative analysis and review where 467 comments were discarded as misclassified from the initially identified 3587 useful comments after the review . Out of the 3120 useful comments 1010 comments were of Constructive Criticism, 1047 comments were of Relevant Information and 1063 comments were of Request Clarification.



Add Column Delete Column Add Row Delete Row Convert...

UsefulComments_Final.csv

Id	PostId	Score	Text	CreationDate	UserId	ContentLicense	Tags	CommentCategory	ReviewedCommentCategory
78205222	45622016	4	ngOnChanges is a lifecycle hook that fires...	8/10/2017 20:20	7176268	CC BY-SA 3.0	<angular>	Relevant Information	Relevant Information
82621117	47829280	4	For future questions: code and error mess...	12/15/2017 9:29	3440745	CC BY-SA 3.0	<c++><intellij-...	Constructive Criticism	Constructive Criticism
95228543	54200824	4	So you recognised the arrow function and...	1/15/2019 14:31	1048572	CC BY-SA 4.0	<javascript><e...	Request Clarification	Request Clarification
110651902	62570219	4	`display(n->left);` but `display` doesn't e...	6/25/2020 7:41	1606345	CC BY-SA 4.0	<c><recursion>	Request Clarification	Request Clarification
111164542	60009473	4	If it is happening only in release builds, th...	7/12/2020 15:17	3484700	CC BY-SA 4.0	<android><an...	Relevant Information	Relevant Information
92632755	52770159	4	@bla I think that as is, its not solvable. Mai...	10/17/2018 15:...	1485872	CC BY-SA 4.0	<image><matl...	Constructive Criticism	Constructive Criticism
90557493	51800249	4	"I ran this through a debugger, but we've...	8/11/2018 14:21	694576	CC BY-SA 4.0	<c><list><seg...	Request Clarification	Request Clarification
80650060	46851298	4	What code do you have in `cellForRowAtIn...	10/20/2017 14:...	2868292	CC BY-SA 3.0	<ios><swift><...	Request Clarification	Request Clarification
108695491	61445787	4	Does this answer your question? [toFixed j...	4/26/2020 18:22	10781526	CC BY-SA 4.0	<javascript><...	Relevant Information	Relevant Information
93855605	187252	4	I'm voting to close this question as off-top...	11/27/2018 4:55	4032703	CC BY-SA 4.0	<wpf><expres...	Constructive Criticism	Constructive Criticism
85409311	49201960	4	Please tag your question with the databas...	3/9/2018 20:56	1144035	CC BY-SA 3.0	<mysql><sql>...	Request Clarification	Request Clarification
91922231	52490715	4	Where is the code that sends vertex data t...	9/25/2018 4:51	734069	CC BY-SA 4.0	<c++><opengl...	Request Clarification	Request Clarification
84688872	48842402	4	Have you overridden hashcode() also? Po...	2/17/2018 14:35	5039498	CC BY-SA 3.0	<java><hashse...	Request Clarification	Request Clarification
78314876	45678593	4	Your code is vulnerable to SQL injection. P...	8/14/2017 16:08	2695799	CC BY-SA 3.0	<php><mysql>	Relevant Information	Relevant Information
96649809	54943573	4	How do you know which words are to be c...	3/1/2019 11:21	8472377	CC BY-SA 4.0	<python><stri...	Request Clarification	Request Clarification
79616724	46327920	4	Your compiler didn't warn about the zero-l...	9/20/2017 17:05	3185968	CC BY-SA 3.0	<c>	Request Clarification	Request Clarification
75093028	44033219	4	Are you looking for a c or a c++ solution?...	5/17/2017 19:28	7359094	CC BY-SA 3.0	<c><algorithm...	Request Clarification	Request Clarification
105858534	59863521	4	Also see https://stackoverflow.com/questi...	1/22/2020 15:56	9245853	CC BY-SA 4.0	<excel><vba>	Relevant Information	Relevant Information
75093438	44033164	4	Admin SDKs have different purpose and c...	5/17/2017 19:40	226391	CC BY-SA 3.0	<node.js><fire...	Relevant Information	Relevant Information
108695912	61446079	4	I suggest you strip this down the least cod...	4/26/2020 18:38	642706	CC BY-SA 4.0	<java><format...	Constructive Criticism	Constructive Criticism
95765535	54479341	4	Please provide sample data and desired re...	2/1/2019 12:17	1144035	CC BY-SA 4.0	<mysql><sql>	Request Clarification	Request Clarification
101169949	57340072	4	It's not clear what your question is. If no n...	8/3/2019 17:53	97337	CC BY-SA 4.0	<cryptography...	Constructive Criticism	Constructive Criticism
105854869	59861658	4	It looks like some of the code is obfuscate...	1/22/2020 14:18	1056460	CC BY-SA 4.0	<java><bots><...	Constructive Criticism	Constructive Criticism
110690640	62592385	4	pls check your data & debug your code, to...	6/26/2020 10:03	150623	CC BY-SA 4.0	<java><date><...	Constructive Criticism	Constructive Criticism
80635874	46844150	4	Post your code and we'll start from there.	10/20/2017 7:24	8707634	CC BY-SA 3.0	<javascript><h...	Request Clarification	Request Clarification
102074754	57822327	4	You should provide some sample input dat...	9/6/2019 12:50	1505169	CC BY-SA 4.0	<php><replace>	Request Clarification	Request Clarification
73909131	43425514	4	Possible duplicate of [How does the "this"...	4/15/2017 11:35	5459839	CC BY-SA 3.0	<reactjs><bab...	Relevant Information	Relevant Information
97610511	55447796	4	For those not familiar with DB2, can you e...	4/1/2019 4:06	10195153	CC BY-SA 4.0	<mysql><db2>	Request Clarification	Request Clarification
71480120	42157459	4	how should the object look like? what hav...	2/10/2017 10:48	1447675	CC BY-SA 3.0	<javascript>	Request Clarification	Request Clarification
109556473	61941522	4	`def main():` is local to `def create_block`...	5/21/2020 19:40	7414759	CC BY-SA 4.0	<python><tkin...	Relevant Information	Relevant Information
94368182	53758043	4	Possible duplicate of [What is the differen...	12/13/2018 8:50	4429015	CC BY-SA 4.0	<javascript><p...	Relevant Information	Relevant Information
91470643	52261480	4	Please fix your formatting.	9/10/2018 15:56	9123163	CC BY-SA 4.0	<python><pyt...	Constructive Criticism	Constructive Criticism
82611089	47824233	4	Are you running PowerShell or your batch...	12/15/2017 0:43	3245749	CC BY-SA 3.0	<powershell><...	Request Clarification	Request Clarification
94368295	53757990	4	@Jordi indeed you should, all we can say i...	12/13/2018 8:53	57695	CC BY-SA 4.0	<java><list><t...	Request Clarification	Request Clarification
97939223	55623166	4	What have you tried so far?	4/11/2019 0:51	3258708	CC BY-SA 4.0	<css><underli...	Request Clarification	Request Clarification

Encoding Unicode (UTF-8) Separator , ; ⇥ | Decimal . , Quote "" \" none Header

Figure 8.2 : Useful Comments which are being labeled after performing qualitative analysis and review

8.2. Reading the data set and TSNE Visualization

Initially the CSV file was read into a dataframe and a visualization was performed using TSNE Visualization to visualize the arrangement of text data in high dimensional space. TSNE is a non-linear method which is used for visualization purposes.

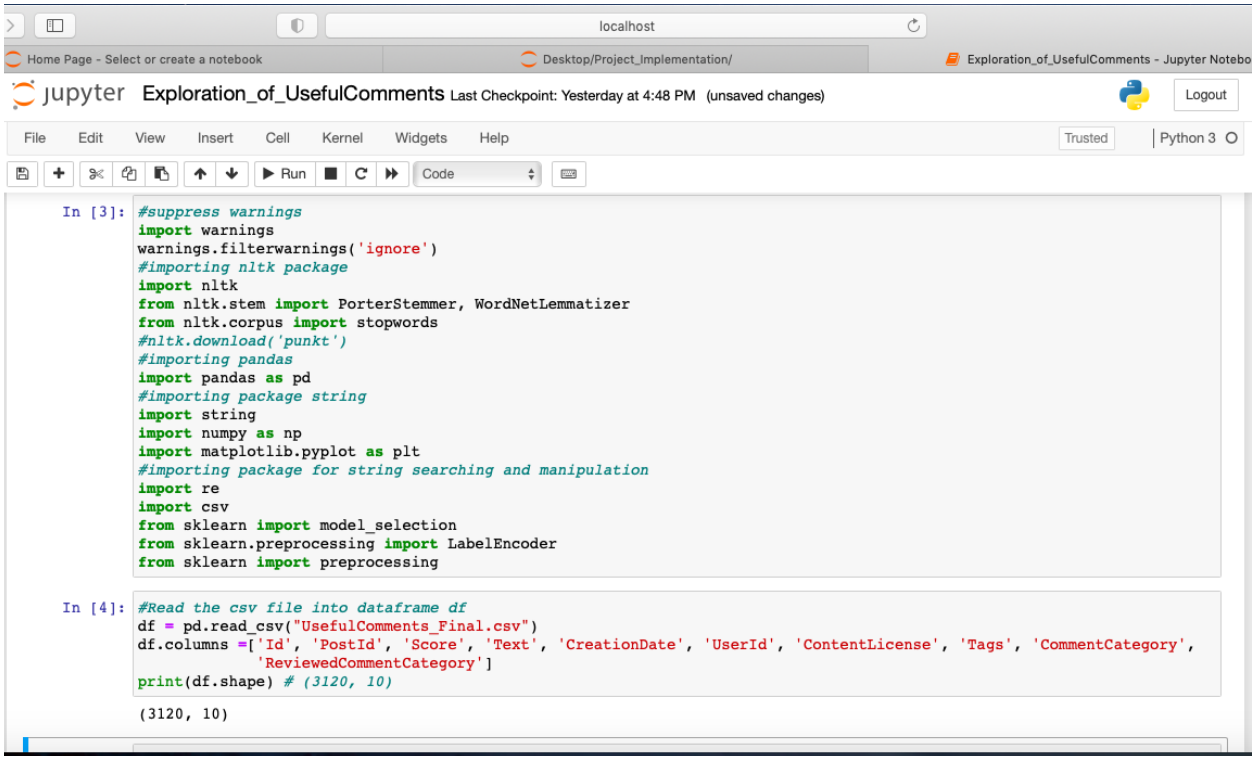


Figure 8.3 : Reading the CSV file into a Dataframe

Figure 8.4 : Listing of fields in the Dataframe

```
# Visualizing the high dimensional comments data using TSNE Visualizer
from sklearn.feature_extraction.text import TfidfVectorizer
from yellowbrick.text import TSNEVisualizer

dataframe = pd.read_csv("UsefulComments_Final.csv")
cmt = dataframe['Text']
lbl = dataframe['ReviewedCommentCategory']

# Load the data and create document vectors
corpus = cmt
tfidf = TfidfVectorizer()

x1 = tfidf.fit_transform(corpus)
y1 = lbl

# Create the visualizer and draw the vectors
tsne = TSNEVisualizer()
tsne.fit(x1, y1)
tsne.show()
```

Figure 8.5 : Visualizing the High Dimensional Comments data using TSNE

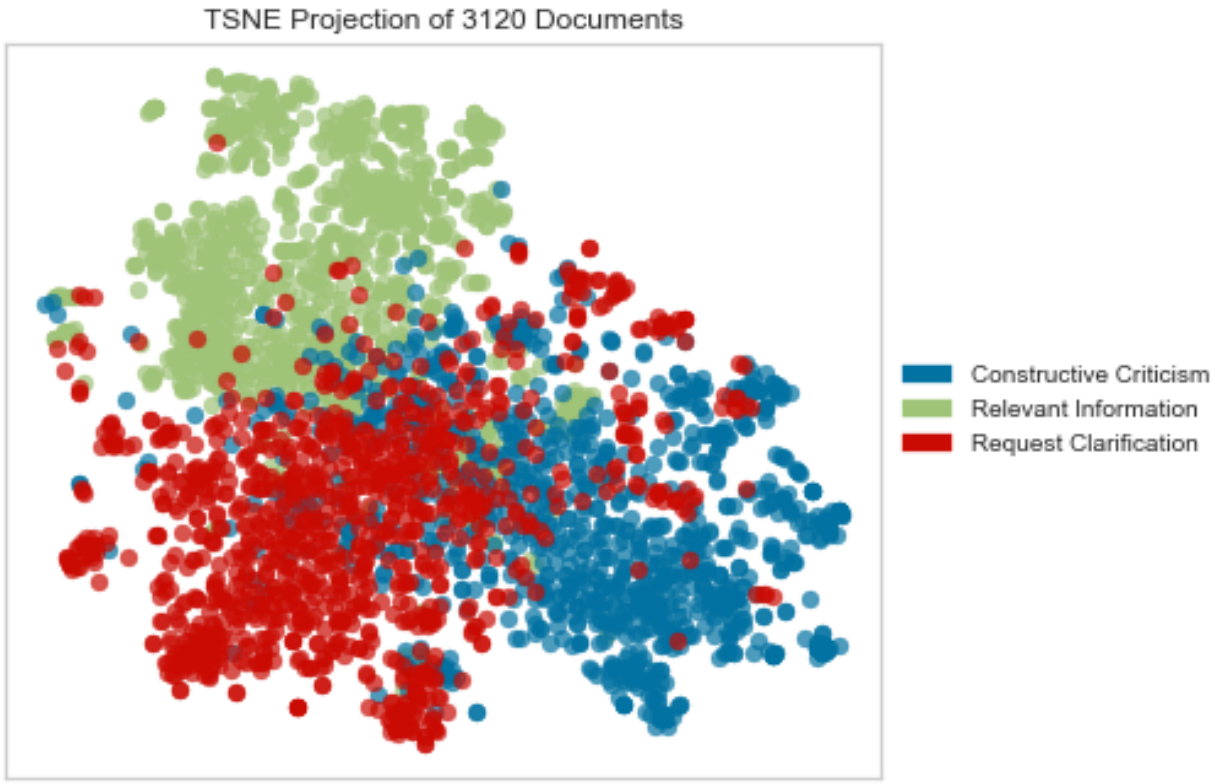


Figure 8.6: TSNE Visualization Plot obtained for 3120 Useful Comments

8.3. Feature Engineering Task with Feature Creation and Feature Evaluation to identify whether new features other than textual features can be added to train the model

Next, a feature engineering trial was performed to identify whether new features can be added to train the classification model. Since there were no prior studies conducted for Useful Comments (Comments which adheres to Commenting Guidelines of Stack Overflow) with Machine Learning this step was important to identify whether new features can be concatenated with text features for the purpose of implementing the Classification Model. This Feature Engineering task included Feature Creation and Feature Evaluation through Histogram Plots. Overlapping of data was clearly identified through horizontal scaling of histograms plotted for all comment categories separately and as a whole. The majority concentration of data points were able to be identified through scatter plots as well. The new features which were created and evaluated were as follows.

1. Comment Length
2. Comment Score
3. Punctuation Percentage
4. Average Word Count
5. Capitalization Usage
6. Stop Words Count
7. Sentiment Score
    a. Positive Sentiment Score
    b. Negative Sentiment Score
    c. Neutral Sentiment Score
    d. Normalized Compound Score

8.3.1. Feature Engineering for Comment Length

The following code segments contain the logic of feature creation, feature evaluation, horizontal scaling for all 3 categories of comments and horizontal scaling for comment categories individually with regard to the Comment Length feature.

```
# Feature Engineering Trial related to Length of Comments
df['character_cnt'] = df['Text'].str.len()
df['word_counts'] = df['Text'].str.split().str.len()

# Feature Engineering for Comment Length
df['comment_length'] = df['Text'].apply(lambda x: len(x))
print(df.head())

from matplotlib import pyplot
import numpy as np

bins = np.linspace(0,650,50)
pyplot.title("Plot related to Feature Evaluation of Comments Length for the entire Comments data")
pyplot.xlabel('Length of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['comment_length'], bins, label =
            'Request Clarification', density = True, color = 'red')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['comment_length'], bins, label =
            'Constructive Criticism', density = True, color = 'green')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['comment_length'], bins, label =
            'Relevant Information', density = True, color = 'blue')

pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Length/Entire_CL.png')
pyplot.show()
```

Figure 8.7 : Feature Creation and Feature Evaluation Related to Comment Length

```
bins = np.linspace(0,200,50)
pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comments Length")
pyplot.xlabel('Length of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['comment_length'], bins, label =
            'Request Clarification', density = True, color = 'red')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['comment_length'], bins, label =
            'Constructive Criticism', density = True, color = 'green')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['comment_length'], bins, label =
            'Relevant Information', density = True, color = 'blue')

pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Length/Scaling_CL.png')
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comments Length for Request Clarificat
pyplot.xlabel('Length of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['comment_length'], bins, label =
            'Request Clarification', density = True, color = 'red')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Length/Request_Clarification_
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comments Length for Constructive Criti
pyplot.xlabel('Length of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['comment_length'], bins, label =
            'Constructive Criticism', density = True, color = 'green')
pyplot.legend(loc='upper right')
```

Figure 8.8 : Horizontal Scaling of Majority Overlapping Area for Comment Length for all comments and comment categories of Request Clarification and Constructive Criticism Separately

```
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Length/Constructive_Criticism
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comments Length for Relevant Informati
pyplot.xlabel('Length of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['comment_length'], bins, label =
            'Relevant Information', density = True, color = 'blue')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Length/Relevant_Information_C
pyplot.show()

df.plot.scatter(x='comment_length',
                      y='ReviewedCommentCategory',
                      c='DarkBlue')
pyplot.title("Scatter plot visualization for Comment Length")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Length/Scatter_CL.png')
```

Figure 8.9 : Horizontal Scaling of Majority Overlapping Area for Relevant Information Comments' Length and Scatter Plot Visualization

The plot related to Feature Evaluation of Comment Length depicts that majority data of Request Clarification and Constructive Criticism overlaps with respect to the Length of Comments.

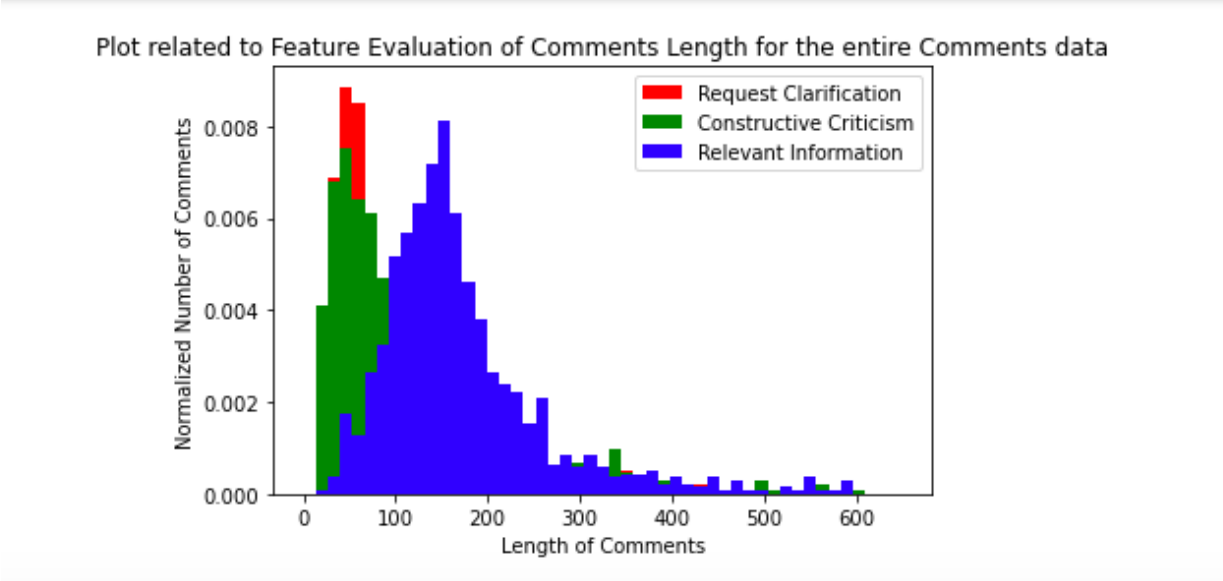


Figure 8.10 : Plot related to Feature Evaluation of Comments Length for the entire Comments data

For the purpose of getting a more clear visualization horizontal scaling for the majority overlapping area was performed as in the plot below. It shows that the majority of Request Clarification Comments and Constructive Criticism Comments overlap with regard to Comment Length as a feature.

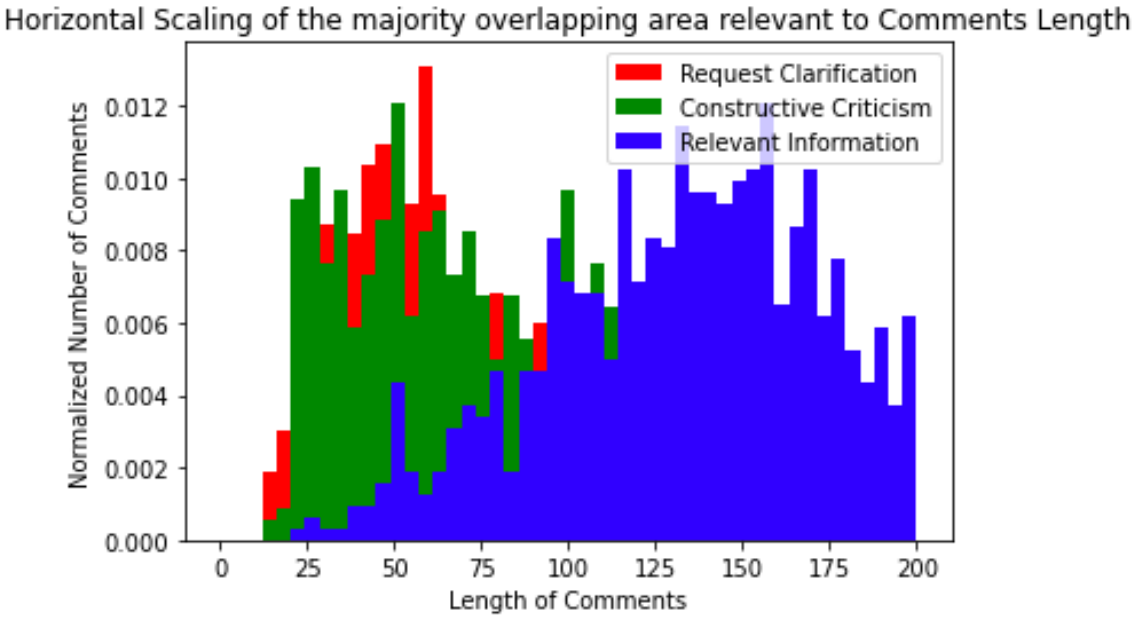


Figure 8.11 : Horizontal Scaling of the majority overlapping area relevant to Comments Length

It was necessary to study the distribution of Comments' Length for each Comment Category individually to identify and make sure the distribution of the majority overlapping area of the aforementioned 2 Comment Categories. Therefore, individual plots of horizontal scaling were depicted for each 3 comment categories as shown below.

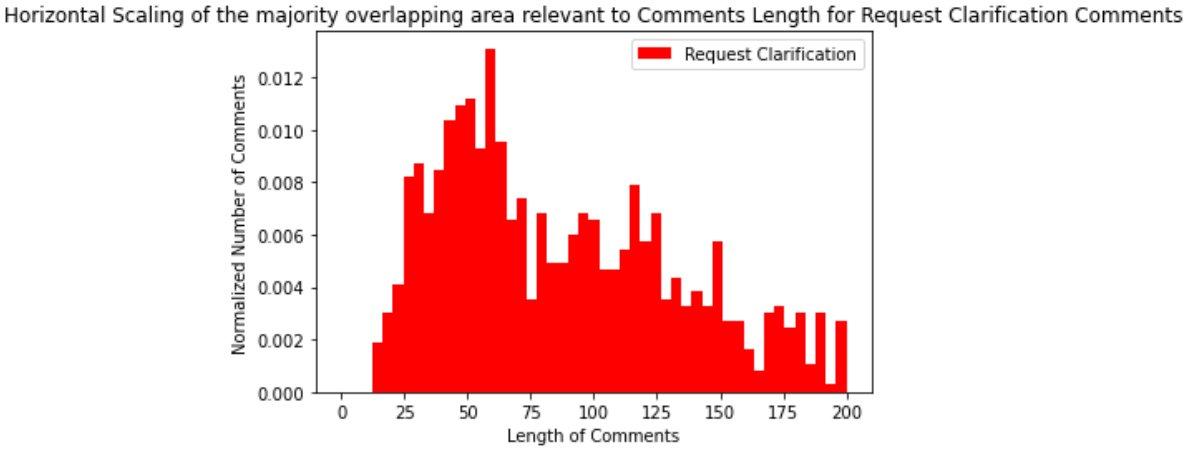


Figure 8.12 : Horizontal Scaling of the majority overlapping area relevant to Comments Length for Request Clarification Comments

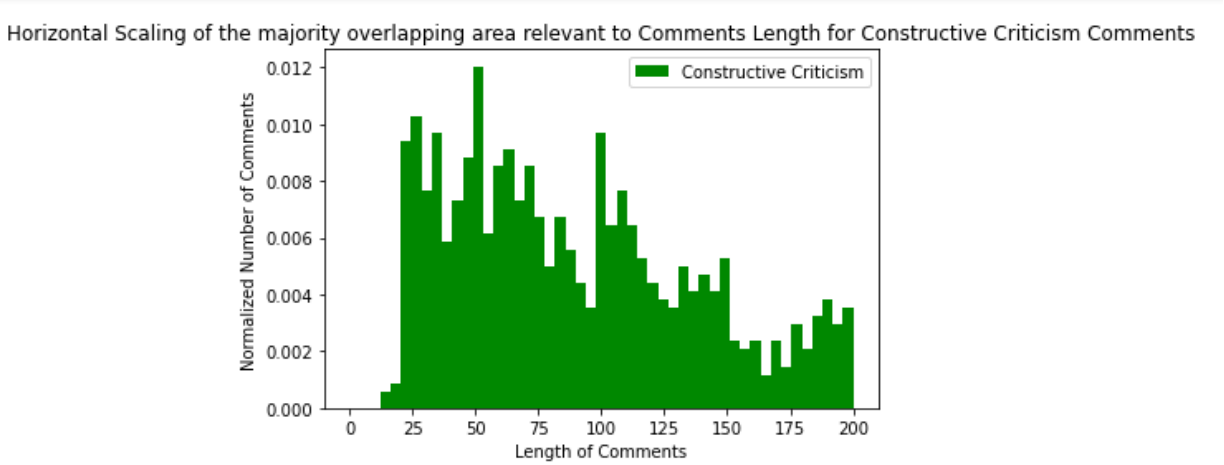


Figure 8.13 : Horizontal Scaling of the majority overlapping area relevant to Comments Length for Constructive Criticism Comments

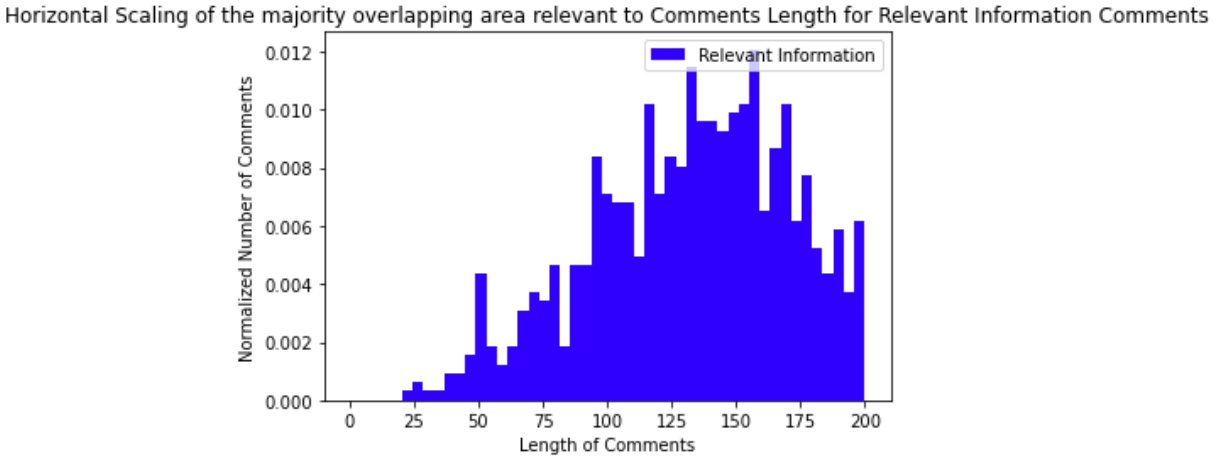


Figure 8.14 : Horizontal Scaling of the majority overlapping area relevant to Comments Length for Relevant Information Comments

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to Length of Comments.

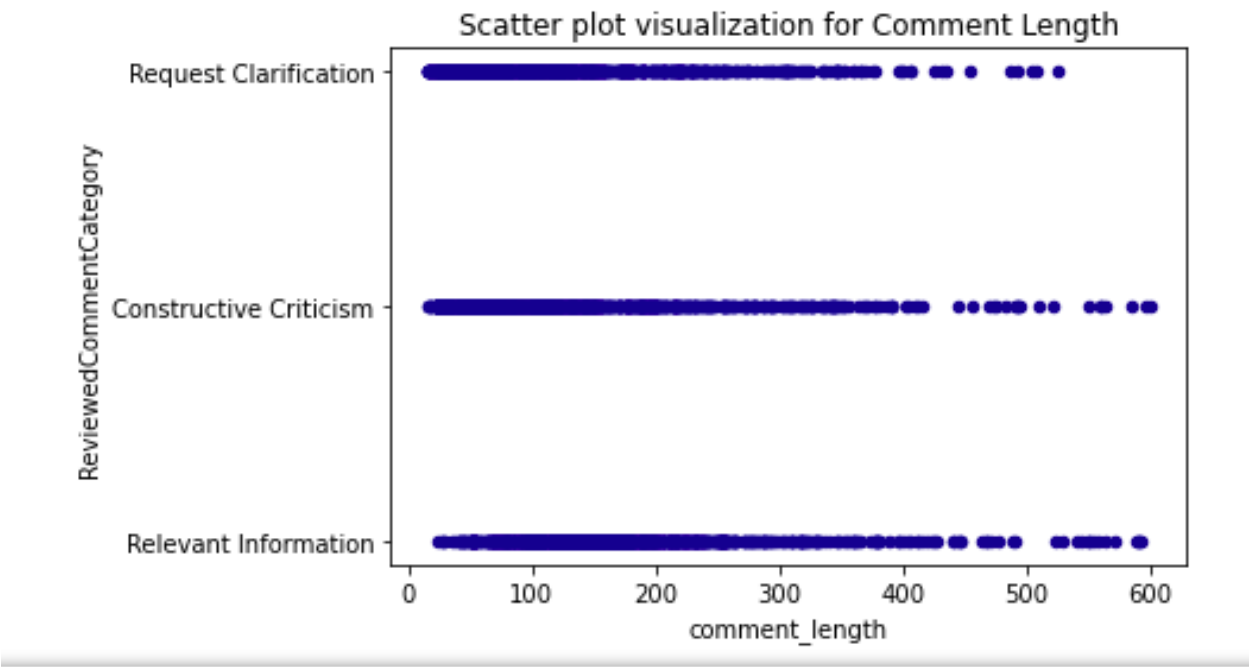


Figure 8.15 : Scatter plot visualization for Comment Length

Since there is a majority overlapping area between 2 Comment Categories, the final decision was to disregard the Length of Comments as a feature during the implementation of the Classification Model.

8.3.2. Feature Engineering for Comment Score

The following code segments contain the logic of feature creation, feature evaluation, horizontal scaling for all 3 categories of comments and horizontal scaling for comment categories individually with regard to the Comment Score feature.

```
# Feature Engineering Trial related to Comment Score

df['comment_score'] = df['Score']
print(df.head())

bins = np.linspace(0,250,50)
pyplot.title("Plot related to Feature Evaluation of Comment Score for the entire Comments data")
pyplot.xlabel('Score of Comments')
pyplot.ylabel('Normalized Number of Comments')

pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['comment_score'], bins, label =
            'Constructive Criticism', density = True, color = 'slateblue')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['comment_score'], bins, label =
            'Request Clarification', density = True, color = 'pink')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['comment_score'], bins, label =
            'Relevant Information', density = True, color = 'cornflowerblue')

pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Score/Entire_CS.png')
pyplot.show()
```

Figure 8.16: Feature Creation and Feature Evaluation Related to Comment Score

```
bins = np.linspace(0,25,50)
pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comment Score")
pyplot.xlabel('Score of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['comment_score'], bins, label =
            'Constructive Criticism', density = True, color = 'slateblue')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['comment_score'], bins, label =
            'Request Clarification', density = True, color = 'pink')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['comment_score'], bins, label =
            'Relevant Information', density = True, color = 'cornflowerblue')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Score/Scaling_CS.png')
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comment Score for Request Clarification C
pyplot.xlabel('Score of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['comment_score'], bins, label =
            'Request Clarification', density = True, color = 'pink')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Score/Request_Clarification_CS.p
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comment Score for Relevant Information Co
pyplot.xlabel('Score of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['comment_score'], bins, label =
            'Relevant Information', density = True, color = 'cornflowerblue')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Score/Relevant_Information_CS.pr
```

Figure 8.17: Horizontal Scaling of Majority Overlapping Area for Comment Score for all comments and comment categories of Request Clarification and Relevant Information Separately

```
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Comment Score for Constructive Criticism
pyplot.xlabel('Score of Comments')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['comment_score'], bins, label =
           'Constructive Criticism', density = True, color = 'slateblue')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Score/Constructive_Criticism_CS.
pyplot.show()

df.plot.scatter(x='comment_score',
                    y='ReviewedCommentCategory',
                    c='Pink')
pyplot.title("Scatter plot visualization for Comment Score")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Comment_Score/Scatter_CS.png')
```

Figure 8.18: Horizontal Scaling of Majority Overlapping Area for Comment Score for Constructive Criticism Comments' Score and Scatter plot Visualization

The plot related to Feature Evaluation of Comment Score depicts that data of Request Clarification, Relevant Information and Constructive Criticism overlaps with respect to the Score of Comments.

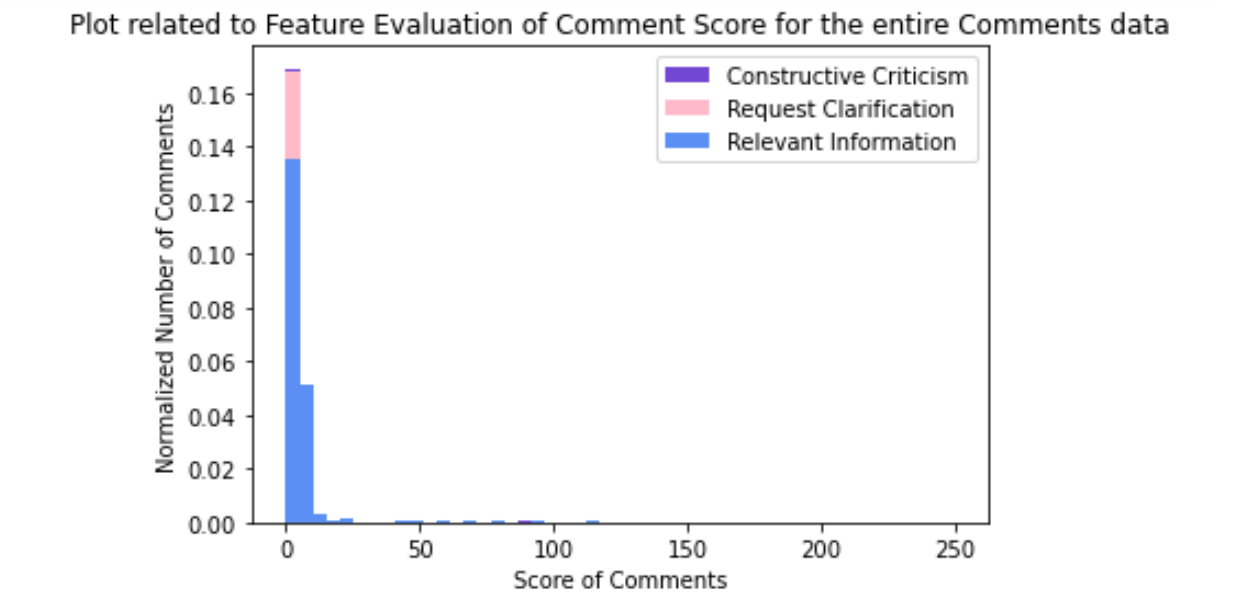


Figure 8.19: Plot related to Feature Evaluation of Comment Score for the entire Comments data

For the purpose of getting a more clear visualization horizontal scaling for the majority overlapping area was performed as in the plot below. It shows that the majority of Request

Clarification Comments and Relevant Information Comments overlap with regard to Comment Score as a feature. Moreover all 3 Comment Categories overlap when the Comment Score is 1.

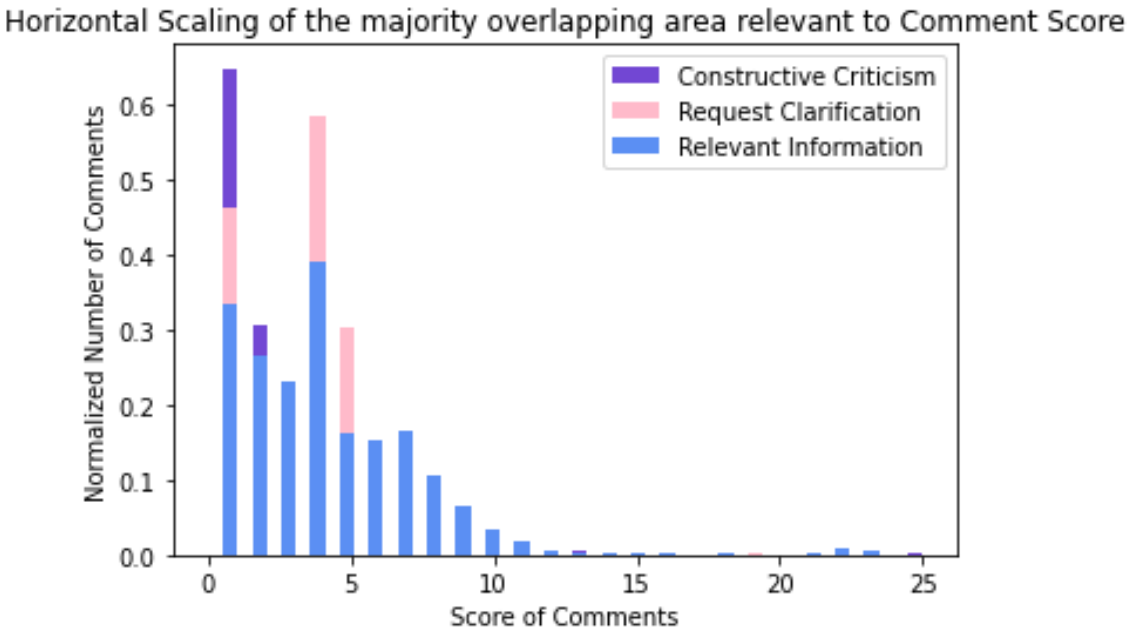


Figure 8.20 : Horizontal Scaling of the majority overlapping area relevant to Comment Score

It was necessary to study the distribution of Comments' Score for each Comment Category individually to identify and make sure the distribution of the majority overlapping area of the aforementioned 3 Comment Categories. Therefore, individual plots of horizontal scaling were depicted for all 3 comment categories separately as shown below.

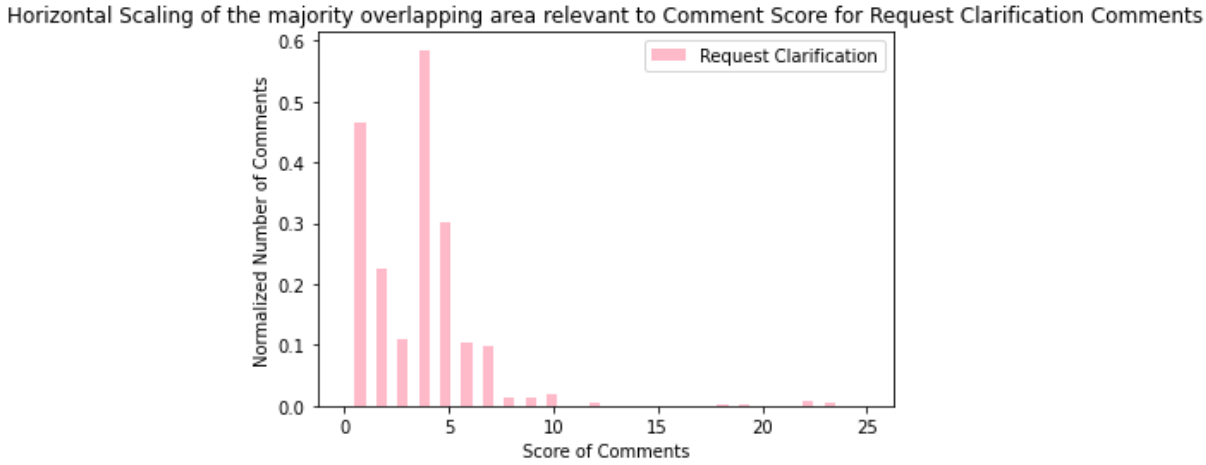


Figure 8.21 : Horizontal Scaling of the majority overlapping area relevant to Comment Score for Request Clarification Comments

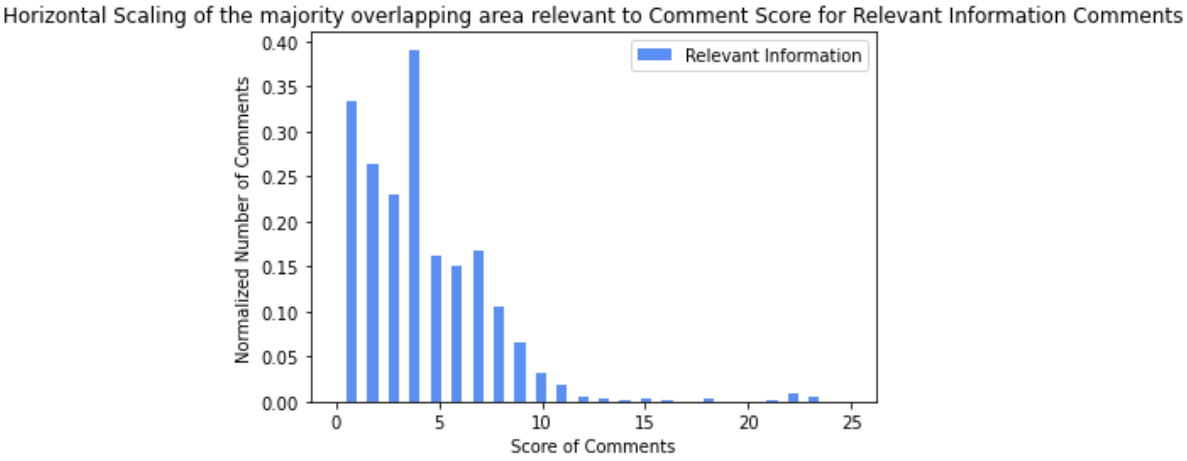


Figure 8.22 : Horizontal Scaling of the majority overlapping area relevant to Comment Score for Relevant Information Comments

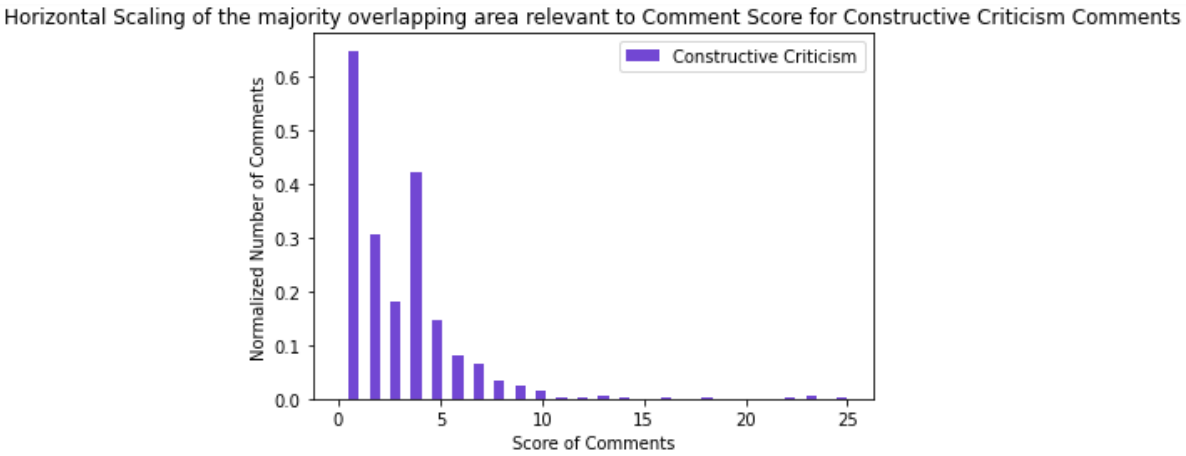


Figure 8.23 : Horizontal Scaling of the majority overlapping area relevant to Comment Score for Constructive Criticism Comments

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to Score of Comments

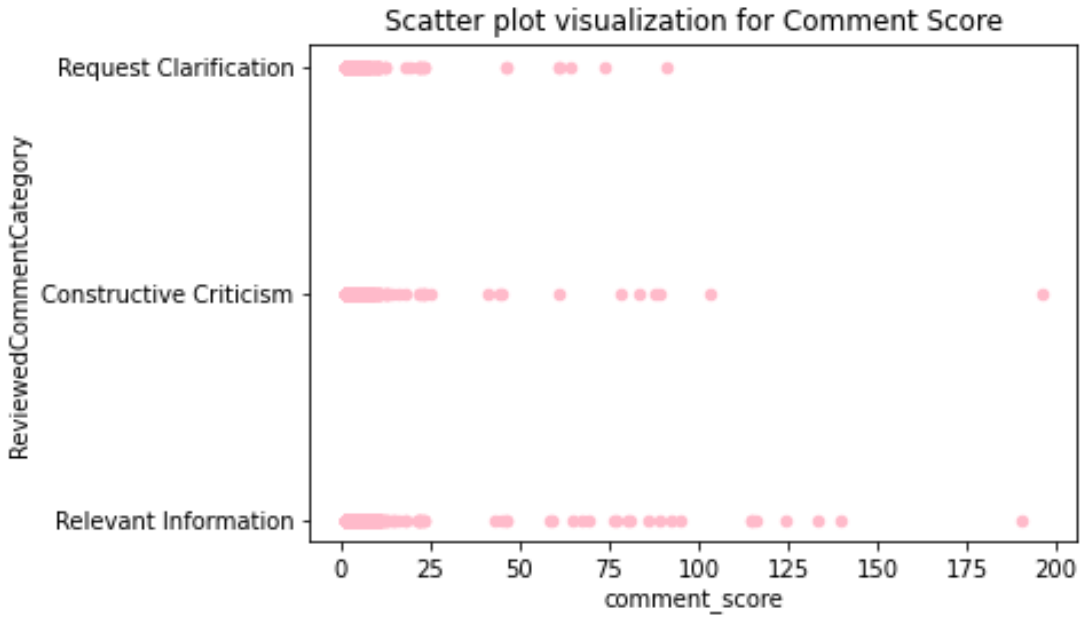


Figure 8.24 : Scatter plot visualization for Comment Score

Since there is a majority overlapping area between all 3 Comment Categories, the final decision was to disregard the Score of Comments as a feature during the implementation of the Classification Model.

8.3.3. Feature Engineering for Punctuation Percentage

The following code segments contain the logic of feature creation, feature evaluation, horizontal scaling for all 3 categories of comments and horizontal scaling for comment categories individually with regard to the Punctuation Percentage feature.

```
# Feature Engineering Trial related to Punctuation Usage

import string
def punctuation_count(text):
    count=sum([1 for c in text if c in string.punctuation])
    return 100*count/len(text)

df['punctuation_%'] = df['Text'].apply(lambda x: punctuation_count(x))
print(df.groupby('ReviewedCommentCategory')['punctuation_%'].mean())

print(df.head())

bins = np.linspace(0,40,50)
pyplot.title("Plot related to Feature Evaluation of Punctuation Percentage for the entire Comments data")
pyplot.xlabel('Punctuation Percentage')
pyplot.ylabel('Normalized Number of Comments')

pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['punctuation_%'], bins, label =
            'Constructive Criticism', density = True, color = 'yellow')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['punctuation_%'], bins, label =
            'Request Clarification', density = True, color = 'orange')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['punctuation_%'], bins, label =
            'Relevant Information', density = True, color = 'brown')

pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Punctuation_Percentage/Entire_PP.png')
pyplot.show()
```

Figure 8.25: Feature Creation and Feature Evaluation Related to Punctuation Percentage

```
bins = np.linspace(0,15,50)
pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage")
pyplot.xlabel('Punctuation Percentage')
pyplot.ylabel('Normalized Number of Comments')

pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['punctuation_%'], bins, label =
            'Constructive Criticism', density = True, color = 'yellow')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['punctuation_%'], bins, label =
            'Request Clarification', density = True, color = 'orange')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['punctuation_%'], bins, label =
            'Relevant Information', density = True, color = 'brown')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Punctuation_Percentage/Scaling_PP.png')
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage for Request Clarif
pyplot.xlabel('Punctuation Percentage')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['punctuation_%'], bins, label =
            'Request Clarification', density = True, color = 'orange')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Punctuation_Percentage/Request_Clarifica
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage for Constructive C
pyplot.xlabel('Punctuation Percentage')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['punctuation_%'], bins, label =
            'Constructive Criticism', density = True, color = 'yellow')
pyplot.legend(loc='upper right')
```

Figure 8.26 : Horizontal Scaling of Majority Overlapping Area for Punctuation Percentage for all comments and comment categories of Request Clarification and Constructive Criticism Separately

```
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Punctuation_Percentage/Constructive_Crit
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage for Relevant Infor
pyplot.xlabel('Punctuation Percentage')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['punctuation_%'], bins, label =
            'Relevant Information', density = True, color = 'brown')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Punctuation_Percentage/Relevant_Informat
pyplot.show()

df.plot.scatter(x='punctuation_%',
                      y='ReviewedCommentCategory',
                      c='Brown')
pyplot.title("Scatter plot visualization for Punctuation Percentage")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Punctuation_Percentage/Scatter_PP.png')
```

Figure 8.27 : Horizontal Scaling of Majority Overlapping Area for Comment Score for Relevant Information Comments' Score and Scatter plot Visualization

The plot related to Feature Evaluation of Punctuation Percentage depicts that majority data of Request Clarification and Constructive Criticism overlaps with respect to the Punctuation Percentage of Comments.

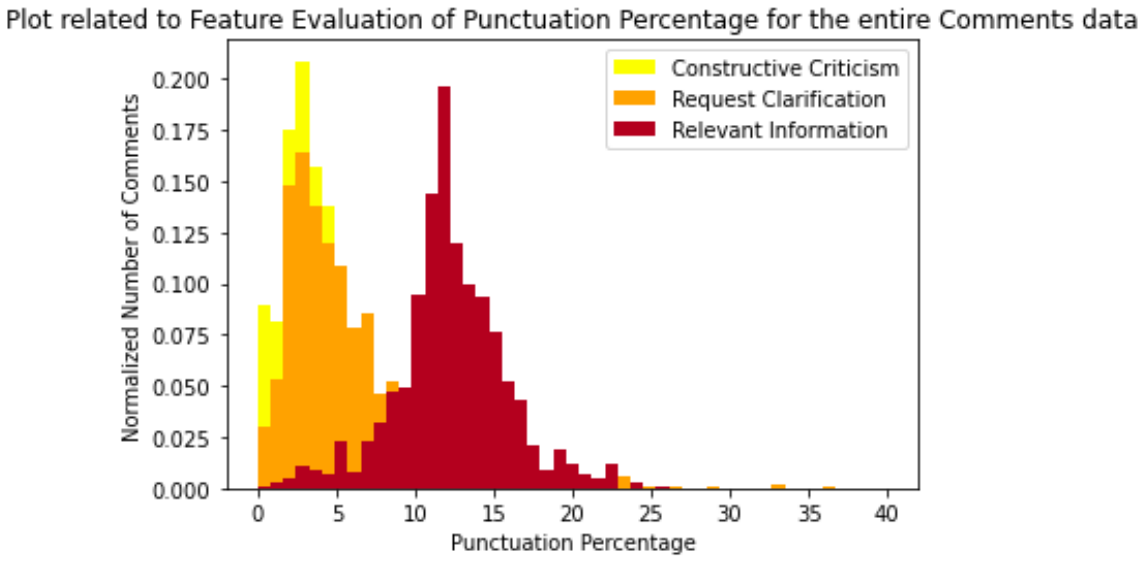


Figure 8.28 : Plot related to Feature Evaluation of Punctuation Percentage for the entire Comments data

For the purpose of getting a more clear visualization horizontal scaling for the majority overlapping area was performed as in the plot below. It shows that the majority of Request Clarification Comments and Constructive Criticism Comments overlap with regard to Punctuation Percentage as a feature.

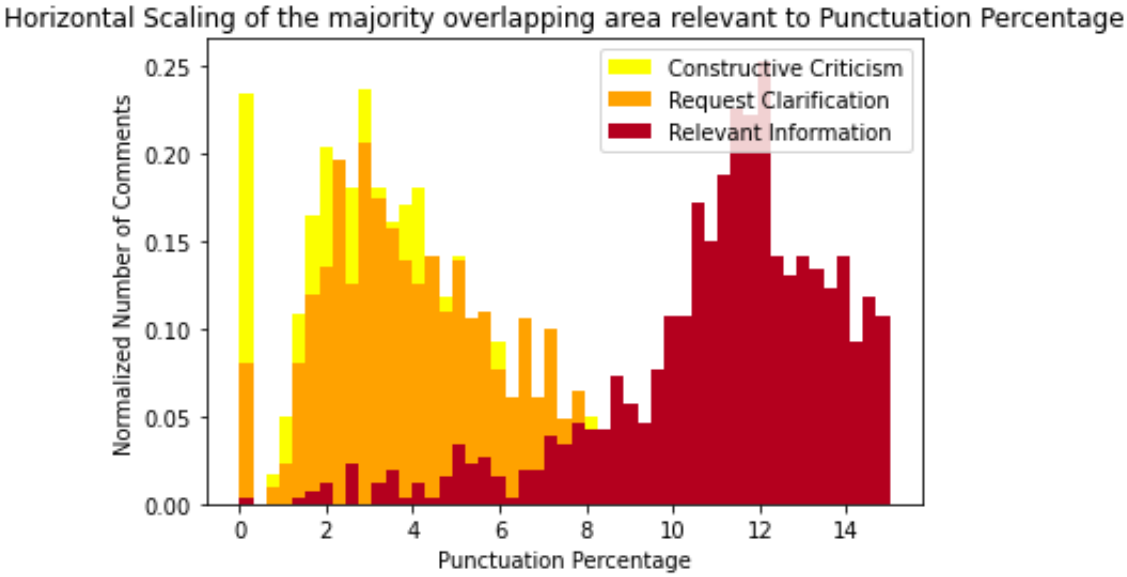


Figure 8.29 : Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage

It was necessary to study the distribution of Comments' Punctuation Percentage for each Comment Category individually to identify and make sure the distribution of the majority overlapping area of the aforementioned 2 Comment Categories. Therefore, individual plots of horizontal scaling were depicted for each 3 comment categories as shown below.

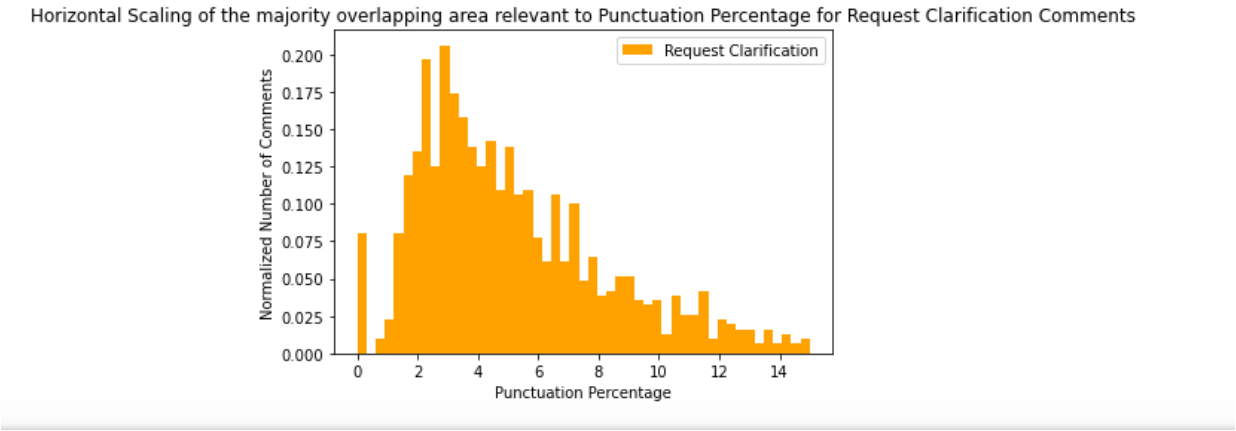


Figure 8.30 : Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage for Request Clarification Comments

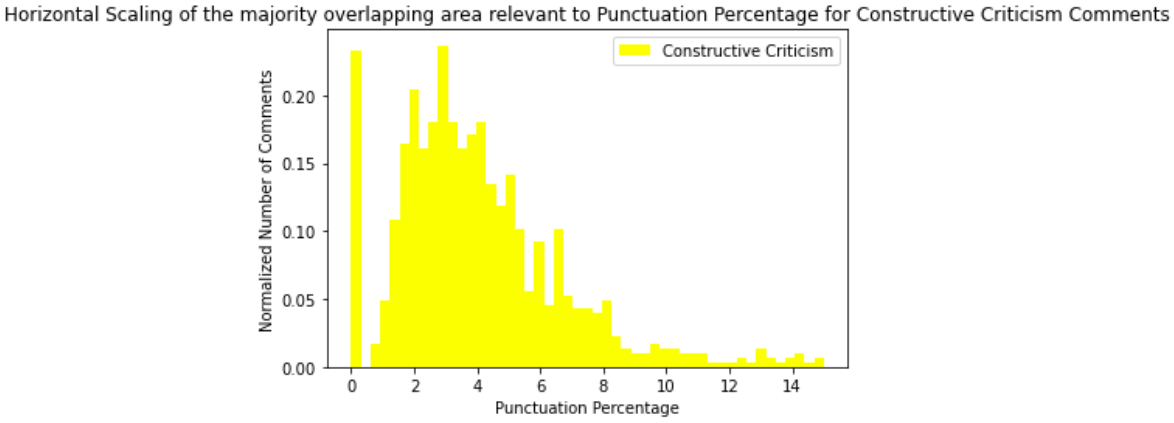


Figure 8.31 : Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage for Constructive Criticism Comments

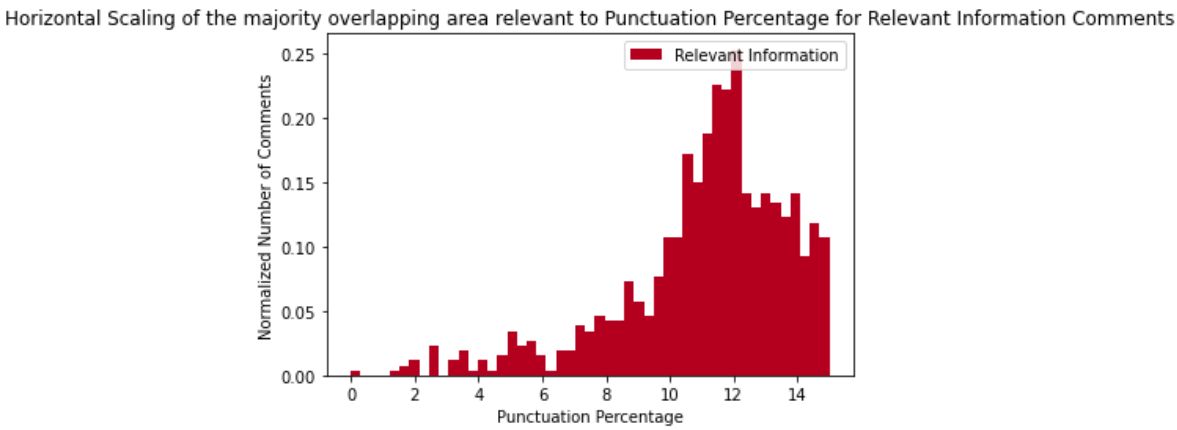


Figure 8.32 : Horizontal Scaling of the majority overlapping area relevant to Punctuation Percentage for Relevant Information Comments

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to Punctuation Percentage of Comments.

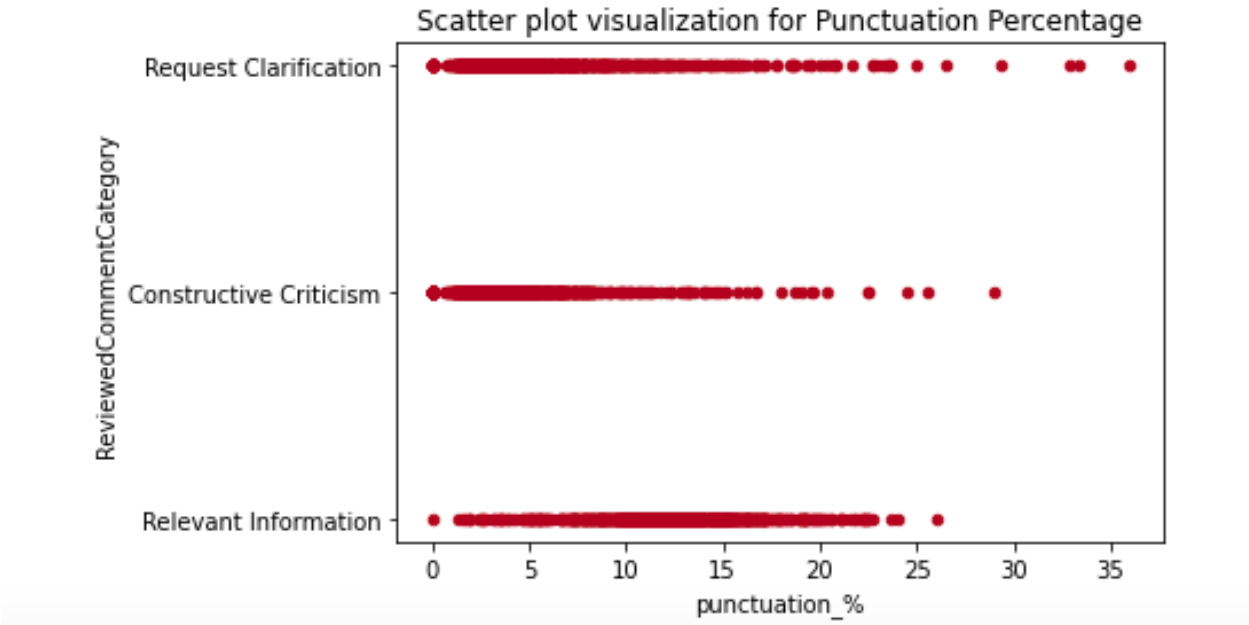


Figure 8.33 : Scatter plot visualization for Punctuation Percentage

Since there is a majority overlapping area between 2 Comment Categories, the final decision was to disregard the Punctuation Percentage of Comments as a feature during the implementation of the Classification Model.

8.3.4. Feature Engineering for Average Word Count

The following code segments contain the logic of feature creation, feature evaluation, horizontal scaling for all 3 categories of comments and horizontal scaling for comment categories individually with regard to the Average Word Count feature.

```
# Feature Engineering Trial related to Average word count

def average_word_count(text):
    words = text.split()
    average = sum(len(word) for word in words) / len(words)
    return average

df['avg_word_count'] = df['Text'].apply(lambda x: average_word_count(x))
print(df.head())

bins = np.linspace(0,140,50)
pyplot.title("Plot related to Feature Evaluation of Average Word Count for the entire Comments data")
pyplot.xlabel('Average Word Count')
pyplot.ylabel('Normalized Number of Comments')

pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['avg_word_count'], bins, label =
            'Request Clarification', density = True, color = 'pink')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['avg_word_count'], bins, label =
            'Constructive Criticism', density = True, color = 'purple')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['avg_word_count'], bins, label =
            'Relevant Information', density = True, color = 'blue')

pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Average_Word_Count/Entire_AWC.png')
pyplot.show()
```

Figure 8.34 : Feature Creation and Feature Evaluation Related to Average Word Count

```
bins = np.linspace(4,10,50)
pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Average Word Count")
pyplot.xlabel('Average Word Count')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['avg_word_count'], bins, label =
            'Request Clarification', density = True, color = 'pink')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['avg_word_count'], bins, label =
            'Constructive Criticism', density = True, color = 'purple')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['avg_word_count'], bins, label =
            'Relevant Information', density = True, color = 'blue')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Average_Word_Count/Scaling_AWC.png')
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Average Word Count for Request Clarifica
pyplot.xlabel('Average Word Count')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['avg_word_count'], bins, label =
            'Request Clarification', density = True, color = 'pink')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Average_Word_Count/Request_Clarificatio
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Average Word Count for Constructive Crit
pyplot.xlabel('Average Word Count')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['avg_word_count'], bins, label =
            'Constructive Criticism', density = True, color = 'purple')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Average_Word_Count/Constructive_Critici
```

Figure 8.35 : Horizontal Scaling of Majority Overlapping Area for Average Word Count for all comments and comment categories of Request Clarification and Constructive Criticism Separately

```
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Average Word Count for Relevant Informati
pyplot.xlabel('Average Word Count')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['avg_word_count'], bins, label =
            'Relevant Information', density = True, color = 'blue')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Average_Word_Count/Relevant_Information_
pyplot.show()

df.plot.scatter(x='avg_word_count',
                y='ReviewedCommentCategory',
                c='Purple')
pyplot.title("Scatter plot visualization for Average Word Count")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Average_Word_Count/Scatter_AWC.png')
```

Figure 8.36 : Horizontal Scaling of Majority Overlapping Area for Average Word Count for Relevant Information Comments' Score and Scatter plot Visualization

The plot related to Feature Evaluation of Average Word Count depicts that majority data of Request Clarification and Constructive Criticism overlaps with respect to the Average Word Count of Comments.

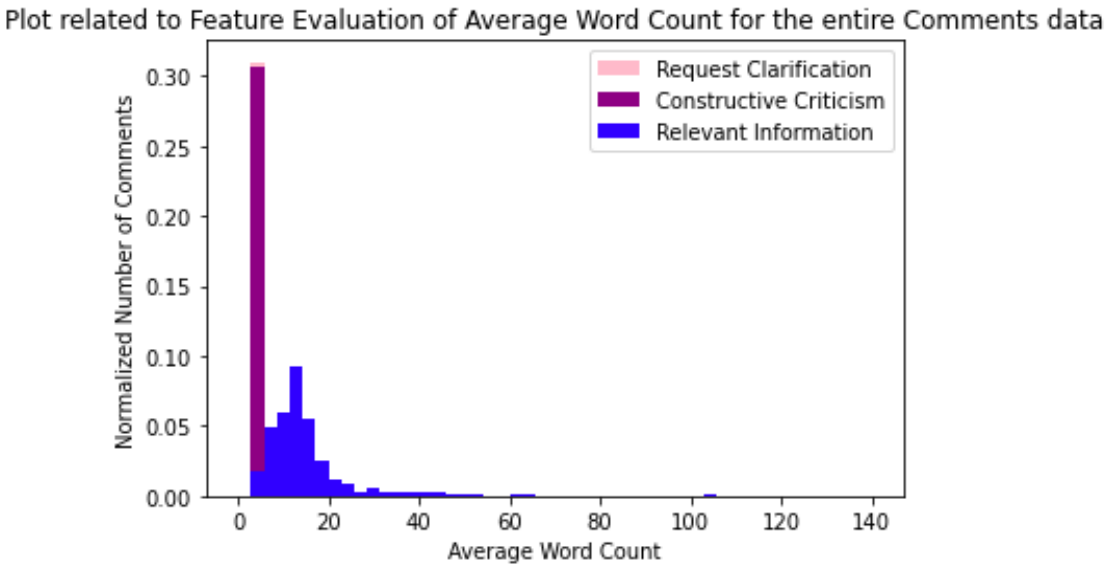


Figure 8.37 : Plot related to Feature Evaluation of Average Word Count for the entire Comments data

For the purpose of getting a more clear visualization horizontal scaling for the majority overlapping area was performed as in the plot below. It shows that the majority of Request Clarification Comments and Constructive Criticism Comments overlap with regard to Average Word Count as a feature.

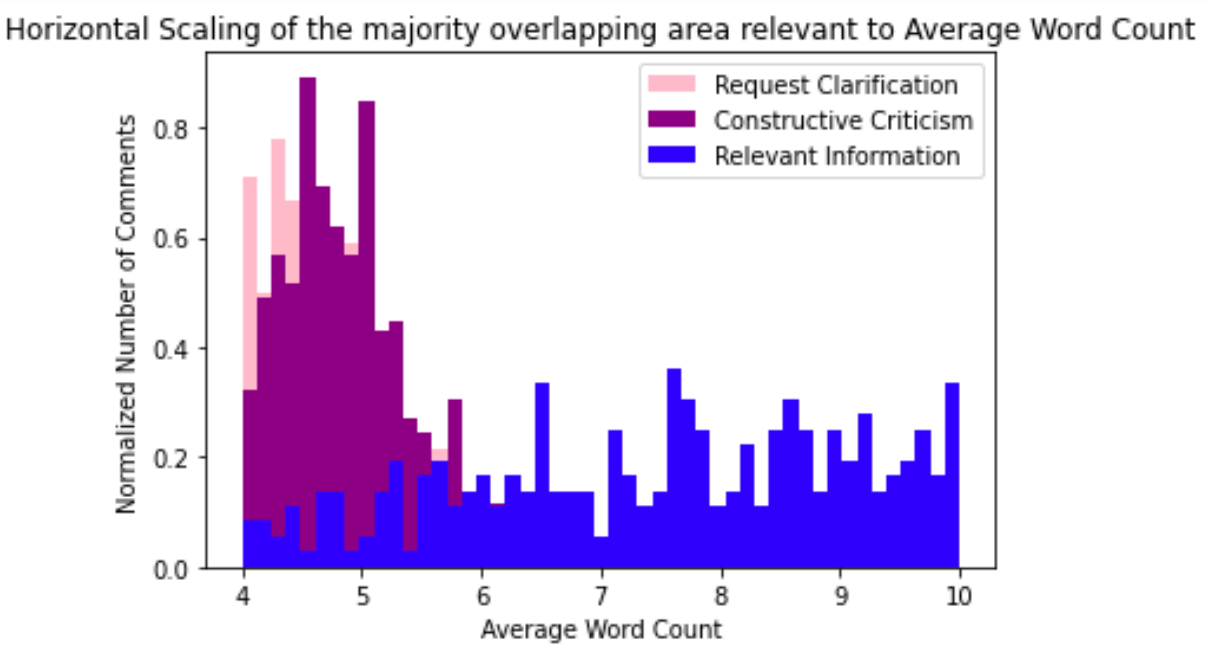


Figure 8.38 : Horizontal Scaling of the majority overlapping area relevant to Average Word Count

It was necessary to study the distribution of Average Word Count for each Comment Category individually to identify and make sure the distribution of the majority overlapping area of the aforementioned 2 Comment Categories. Therefore, individual plots of horizontal scaling were depicted for each 3 comment categories as shown below.

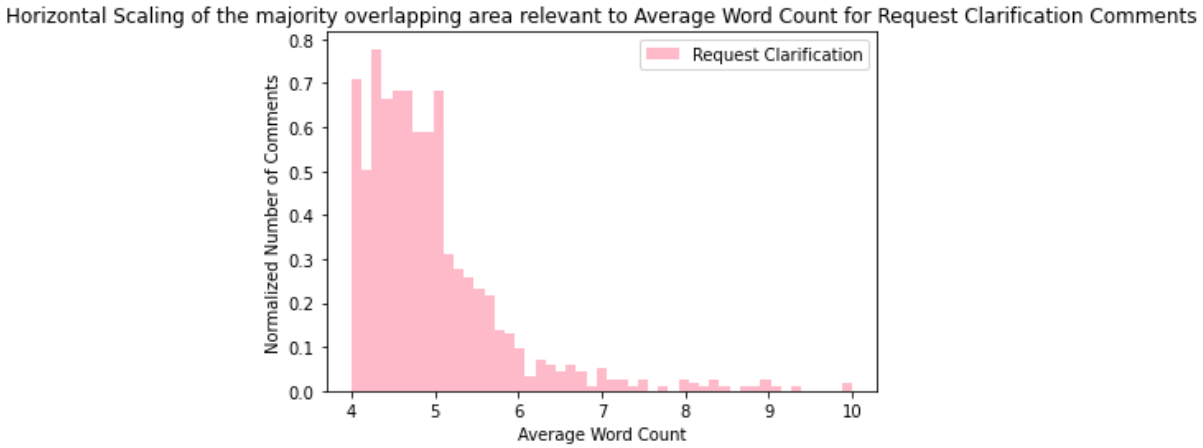


Figure 8.39 : Horizontal Scaling of the majority overlapping area relevant to Average Word Count for Request Clarification Comments

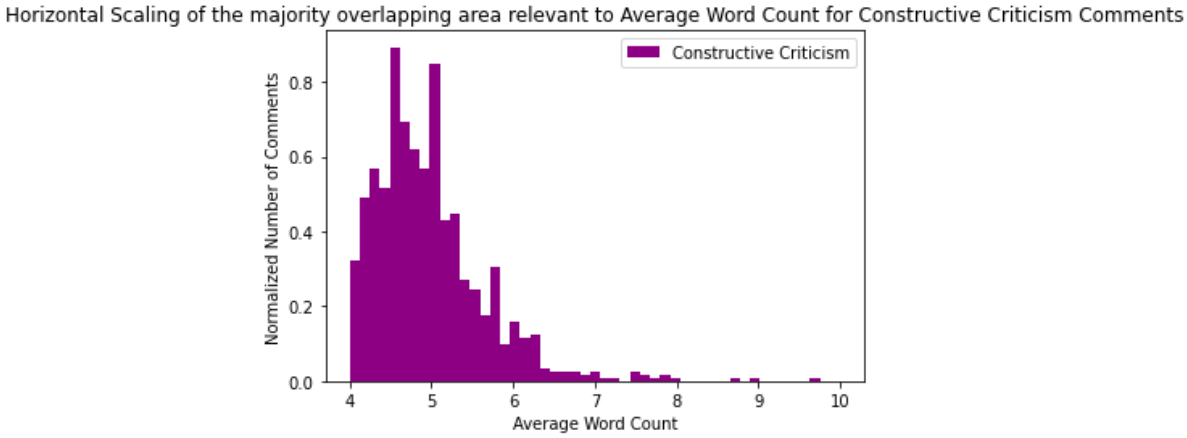


Figure 8.40 : Horizontal Scaling of the majority overlapping area relevant to Average Word Count for Constructive Criticism Comments

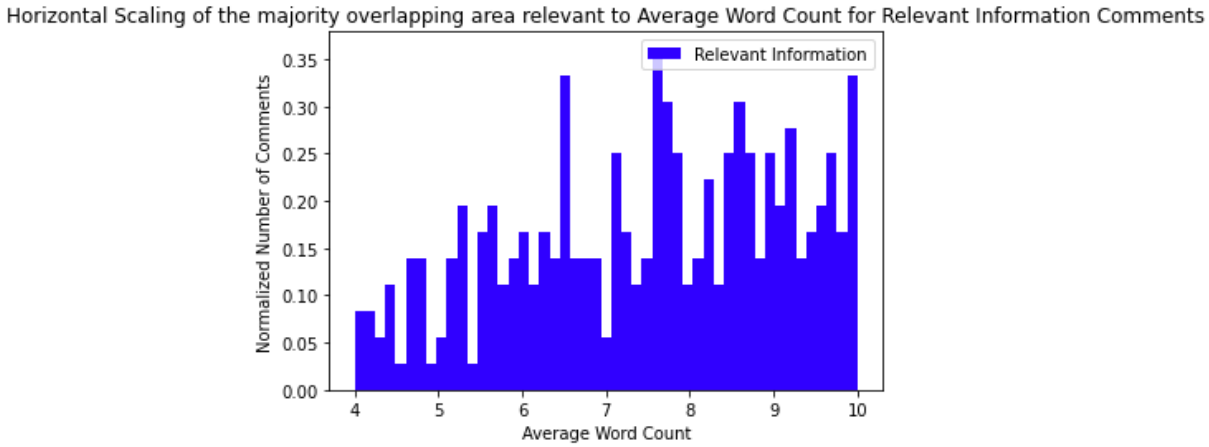


Figure 8.41 : Horizontal Scaling of the majority overlapping area relevant to Average Word Count for Relevant Information Comments

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to Average Word Count of Comments.

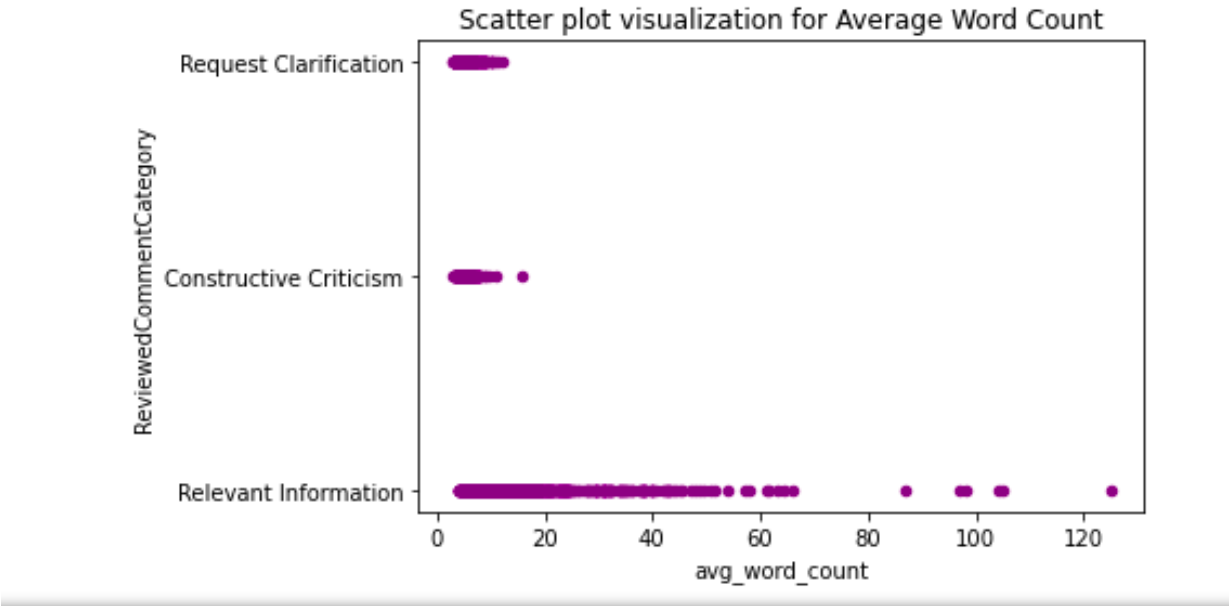


Figure 8.42 : Scatter plot visualization for Average Word Count

Since there is a majority overlapping area between 2 Comment Categories, the final decision was to disregard the Average Word Count as a feature during the implementation of the Classification Model.

8.3.5. Feature Engineering for Capitalization Usage

The following code segments contain the logic of feature creation, feature evaluation, horizontal scaling for all 3 categories of comments and horizontal scaling for comment categories individually with regard to the Capitalization Usage feature. In this feature the Count of Capital Letter of each Comment would be taken into Consideration.

```
# Feature Engineering Trial related to Capitalization Usage in Comments

def capital_count(text):
    num_capital = sum(1 for c in text if c.isupper())
    return num_capital

df['capital_count'] = df['Text'].apply(lambda x: capital_count(x))
print(df.head())

bins = np.linspace(0,75,50)
pyplot.title("Plot related to Feature Evaluation of Capitalization Usage for the entire Comments data")
pyplot.xlabel('Count of Capital Letters')
pyplot.ylabel('Normalized Number of Comments')

pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['capital_count'], bins, label =
            'Constructive Criticism', density = True, color = 'brown')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['capital_count'], bins, label =
            'Request Clarification', density = True, color = 'olive')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['capital_count'], bins, label =
            'Relevant Information', density = True, color = 'cyan')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Capitalization_Usage/Entire_CU.png')
pyplot.show()
```

Figure 8.43 : Feature Creation and Feature Evaluation Related to Capitalization Usage

```
bins = np.linspace(0,10,50)
pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage")
pyplot.xlabel('Count of Capital Letters')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['capital_count'], bins, label =
            'Constructive Criticism', density = True, color = 'brown')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['capital_count'], bins, label =
            'Request Clarification', density = True, color = 'olive')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['capital_count'], bins, label =
            'Relevant Information', density = True, color = 'cyan')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Capitalization_Usage/Scaling_CU.png')
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage for Request Clarific
pyplot.xlabel('Count of Capital Letters')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['capital_count'], bins, label =
            'Request Clarification', density = True, color = 'olive')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Capitalization_Usage/Request_Clarificati
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage for Constructive Cri
pyplot.xlabel('Count of Capital Letters')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['capital_count'], bins, label =
            'Constructive Criticism', density = True, color = 'brown')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Capitalization_Usage/Constructive_Critic
```

Figure 8.44 : Horizontal Scaling of Majority Overlapping Area for Capitalization Usage for all comments and comment categories of Request Clarification and Constructive Criticism Separately

```
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage for Relevant Informa
pyplot.xlabel('Count of Capital Letters')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['capital_count'], bins, label =
            'Relevant Information', density = True, color = 'cyan')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Capitalization_Usage/Relevant_Informatic
pyplot.show()

df.plot.scatter(x='capital_count',
                      y='ReviewedCommentCategory',
                      c='Red')
pyplot.title("Scatter plot visualization for Capitalization Usage")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Capitalization_Usage/Scatter_CU.png')
```

Figure 8.45 : Horizontal Scaling of Majority Overlapping Area for Capitalization Usage for Relevant Information Comments' Score and Scatter plot Visualization

The plot related to Feature Evaluation of Capitalization Usage depicts that majority data of Request Clarification, Relevant Information and Constructive Criticism overlaps with respect to the Capitalization Usage of Comments.

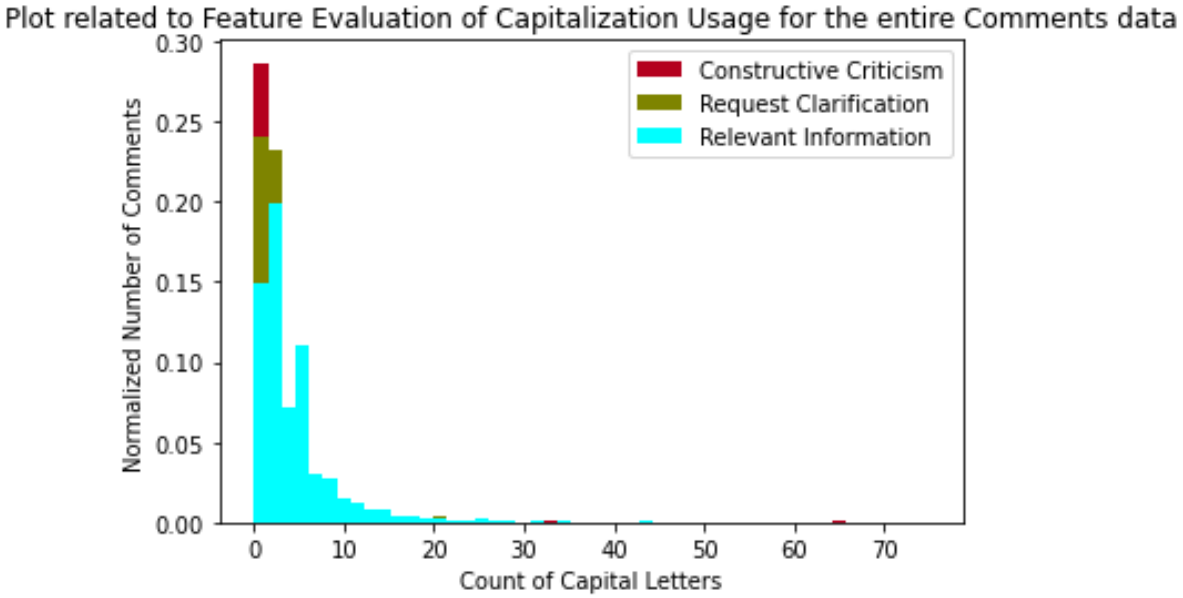


Figure 8.46 : Plot related to Feature Evaluation of Capitalization Usage for the entire Comments data

For the purpose of getting a more clear visualization horizontal scaling for the majority overlapping area was performed as in the plot below. It shows that the majority of Request

Clarification Comments, Relevant Information Comments and Constructive Criticism Comments overlap with regard to Capitalization Usage as a feature.

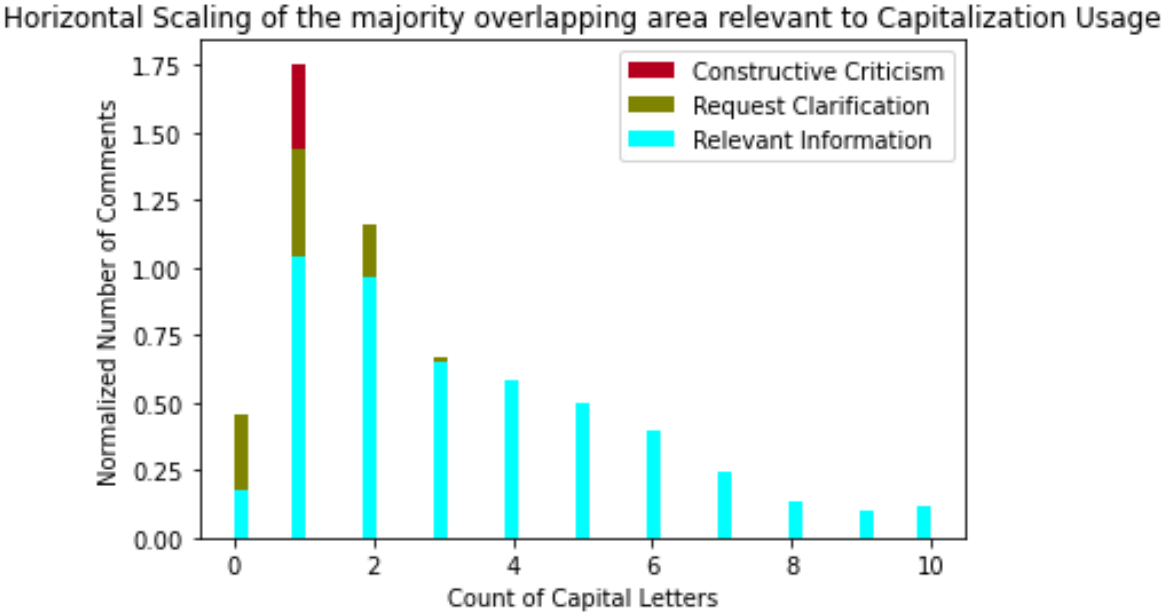


Figure 8.47 : Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage

It was necessary to study the distribution of Comments' Capitalization Usage for each Comment Category individually to identify and make sure the distribution of the majority overlapping area of the all 3 Comment Categories. Therefore, individual plots of horizontal scaling were depicted for each 3 comment categories as shown below.

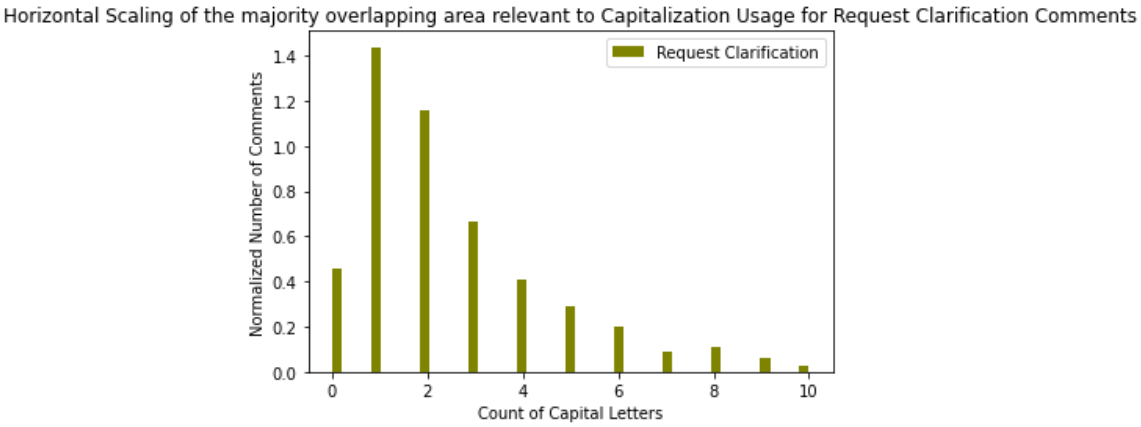


Figure 8.48 : Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage for Request Clarification Comments

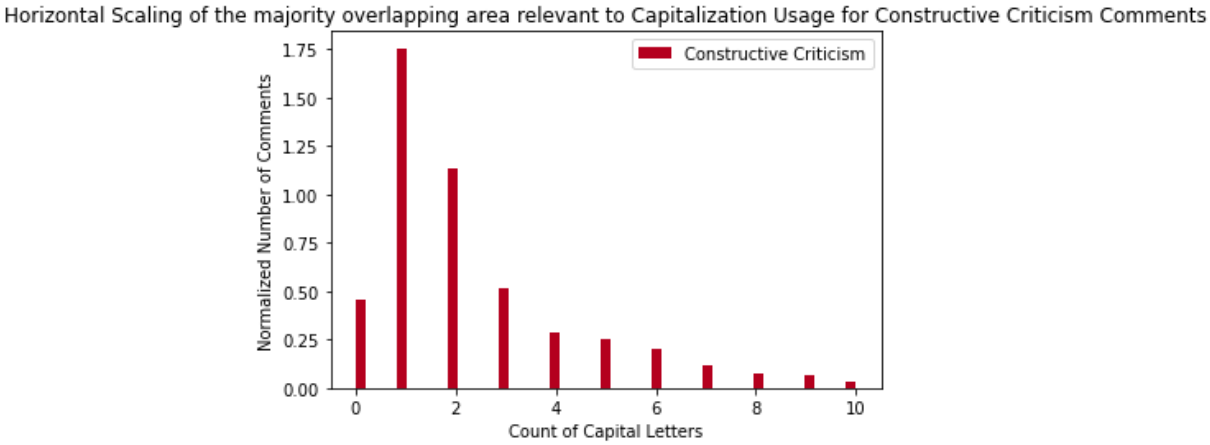


Figure 8.49 : Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage for Constructive Criticism Comments

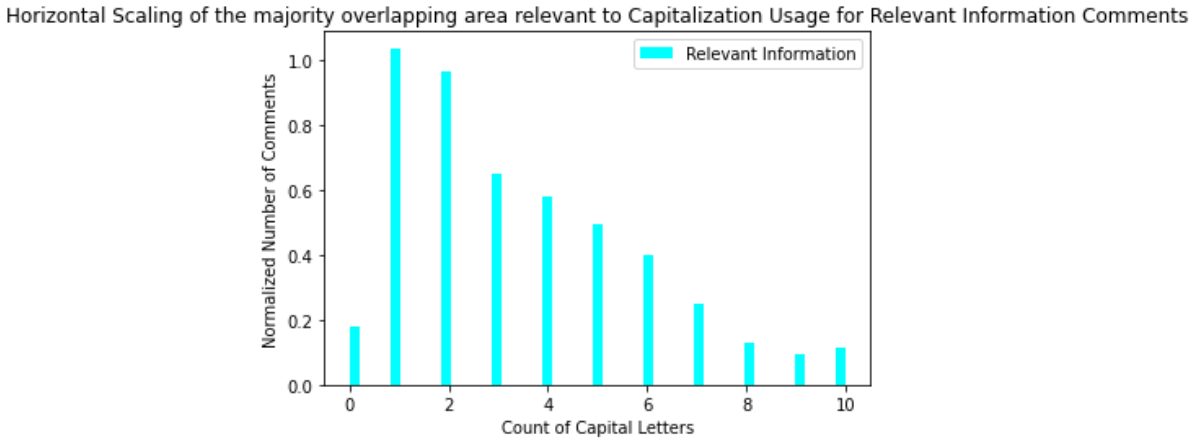


Figure 8.50 : Horizontal Scaling of the majority overlapping area relevant to Capitalization Usage for Relevant Information Comments

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to Capitalization Usage of Comments.

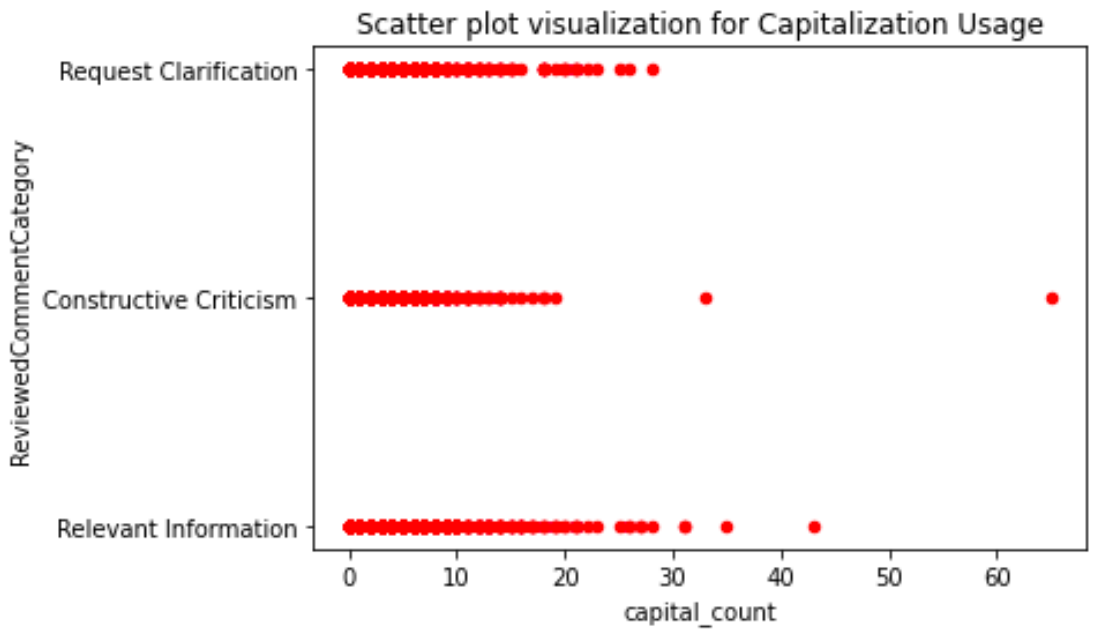


Figure 8.51 : Scatter plot visualization for Capitalization Usage

Since there is a majority overlapping area between all 3 Comment Categories, the final decision was to disregard the Capitalization Usage of Comments as a feature during the implementation of the Classification Model.

8.3.6. Feature Engineering for Stop Words Usage

The following code segment was used to update the list of stopwords where each individual character of the alphabet was appended to the list of Stop Words.

```
from nltk.corpus import stopwords
# Updating the List of Stop Words
nltk.download('stopwords')
stopwords = stopwords.words('english')
stopwords.append('')
for x in range(ord('b'), ord('z')+1):
    stopwords.append(chr(x))
print(stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', '
yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it'
s", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
"that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'a
t', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'bel
ow', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here
', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'suc
h', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "
don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't", '', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/prasadhiranasinghe/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Figure 8.52 : Updating the list of Stop Words

The following code segments contain the logic of feature creation, feature evaluation, horizontal scaling for all 3 categories of comments and horizontal scaling for comment categories individually with regard to the Stop Words Usage feature.

```
# Feature Engineering Trial related to Stop Words Usage in Comments

from nltk.tokenize import word_tokenize

def stopword_count(text):
    stop_word_count = 0
    word_tokens=word_tokenize(text)
    for word in word_tokens:
            if word in stopwords:
                stop_word_count += 1
    return stop_word_count

df['stopword_count'] = df['Text'].apply(lambda x: stopword_count(x))
print(df.head())

bins = np.linspace(0,75,50)
pyplot.title("Plot related to Feature Evaluation of Stop Words Usage for the entire Comments data")
pyplot.xlabel('Count of Stop Words')
pyplot.ylabel('Normalized Number of Comments')

pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['stopword_count'], bins, label =
            'Constructive Criticism', density = True, color = 'lime')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['stopword_count'], bins, label =
            'Relevant Information', density = True, color = 'plum')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['stopword_count'], bins, label =
            'Request Clarification', density = True, color = 'salmon')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/StopWords_Usage/Entire_SWU.png')
pyplot.show()
```

Figure 8.53 : Feature Creation and Feature Evaluation Related to Stop Words Usage

```
bins = np.linspace(0,10,50)
pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage")
pyplot.xlabel('Count of Stop Words')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['stopword_count'], bins, label =
            'Constructive Criticism', density = True, color = 'lime')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['stopword_count'], bins, label =
            'Relevant Information', density = True, color = 'plum')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['stopword_count'], bins, label =
            'Request Clarification', density = True, color = 'salmon')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/StopWords_Usage/Scaling_SWU.png')
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage for Relevant Information
pyplot.xlabel('Count of Stop Words')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['stopword_count'], bins, label =
            'Relevant Information', density = True, color = 'plum')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/StopWords_Usage/Relevant_Information_SWU
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage for Constructive Critici
pyplot.xlabel('Count of Stop Words')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['stopword_count'], bins, label =
            'Constructive Criticism', density = True, color = 'lime')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/StopWords_Usage/Constructive_Criticism_S
```

Figure 8.54 : Horizontal Scaling of Majority Overlapping Area for Stop Words Usage for all comments and comment categories of Relevant Information and Constructive Criticism Separately

```
pyplot.show()

pyplot.title("Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage for Request Clarificatic
pyplot.xlabel('Count of Stop Words')
pyplot.ylabel('Normalized Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['stopword_count'], bins, label =
            'Request Clarification', density = True, color = 'salmon')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/StopWords_Usage/Request_Clarification_SW
pyplot.show()

df.plot.scatter(x='stopword_count',
                      y='ReviewedCommentCategory',
                      c='Black')
pyplot.title("Scatter plot visualization for Stop Words Usage")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/StopWords_Usage/Scatter_SWU.png')
```

Figure 8.55 : Horizontal Scaling of Majority Overlapping Area for Stop Words Usage for Request Clarification Comments' Score and Scatter plot Visualization

The plot related to Feature Evaluation of Stop Words Usage depicts that majority data of Request Clarification and Relevant Information overlaps with respect to the Stop Words Usage.

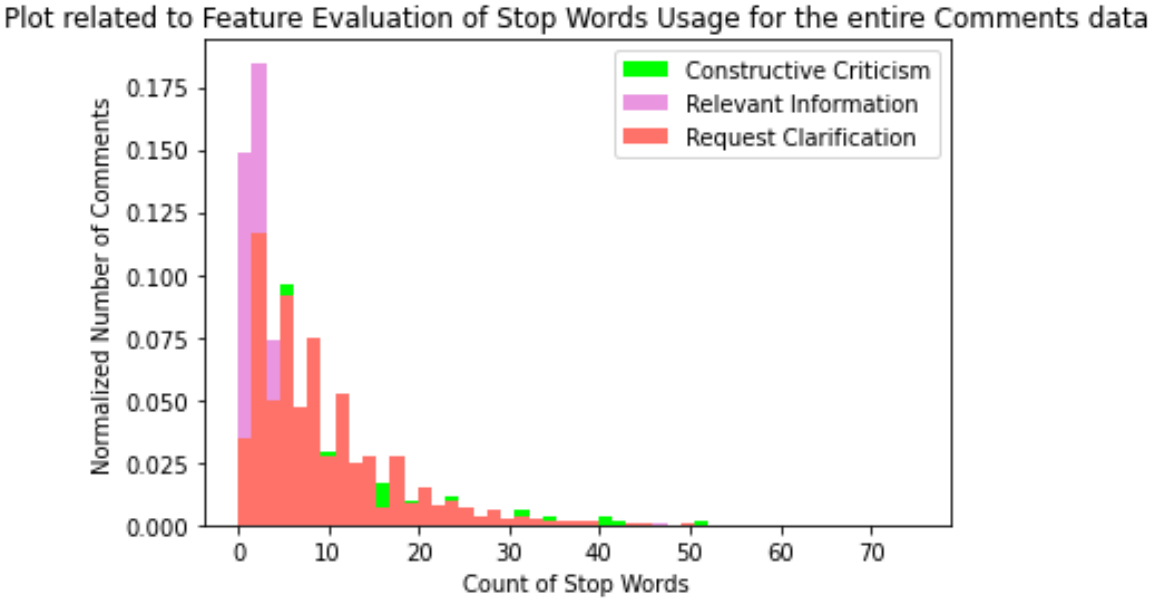


Figure 8.56 : Plot related to Feature Evaluation of Stop Words Usage for the entire Comments data

For the purpose of getting a more clear visualization horizontal scaling for the majority overlapping area was performed as in the plot below. It shows that the majority of Request Clarification Comments and Relevant Information Comments overlap with regard to Stop Words

Usage as a feature. Moreover, when the Stop Words Count is 1, there is an overlap between all 3 Comment Categories' data.

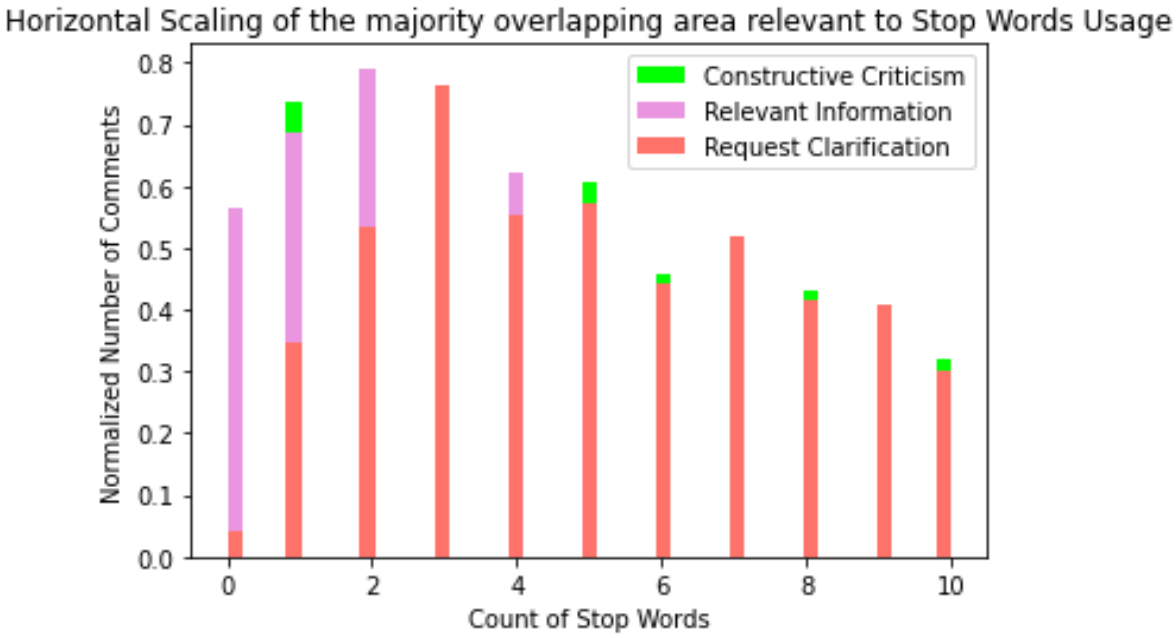


Figure 8.57 : Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage

It was necessary to study the distribution of Comments' Stop Words Usage for each Comment Category individually to identify and make sure the distribution of the majority overlapping area of the aforementioned 2 Comment Categories. Therefore, individual plots of horizontal scaling were depicted for each 3 comment categories as shown below.

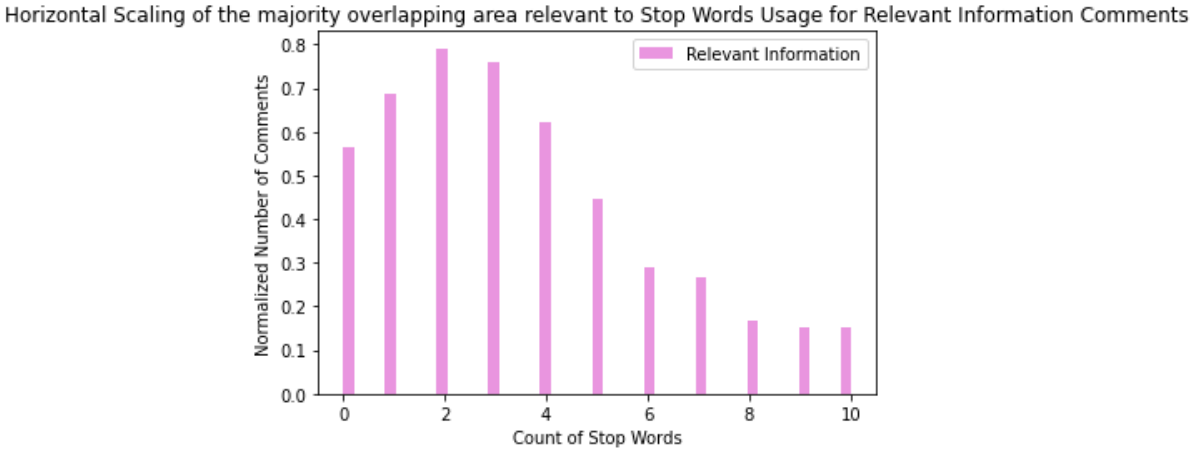


Figure 8.58 : Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage for Relevant Information Comments

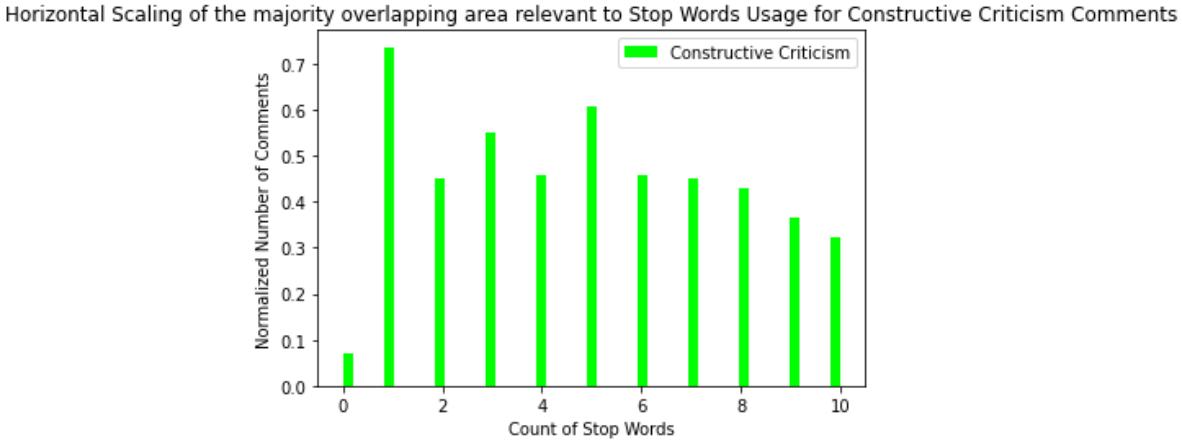


Figure 8.59 : Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage for Constructive Criticism Comments

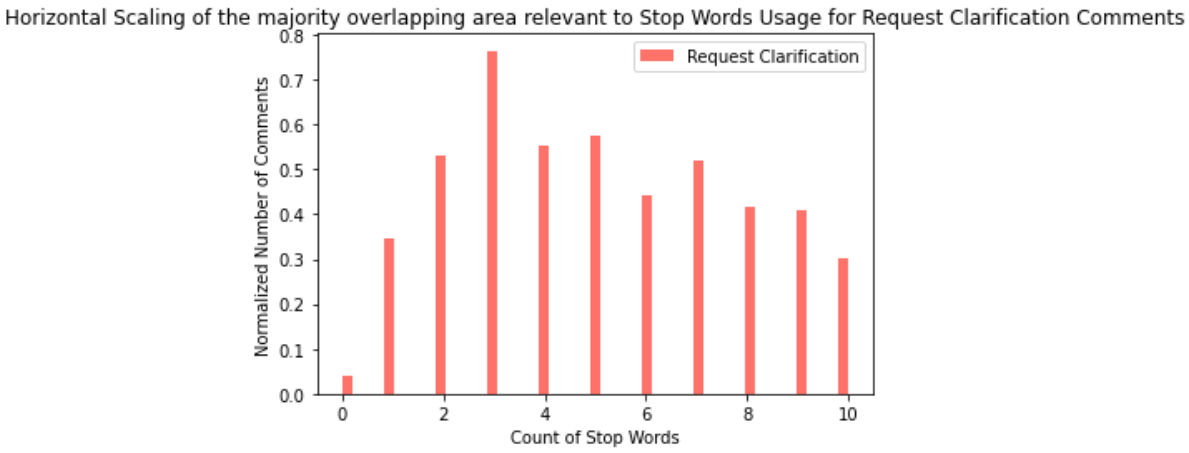


Figure 8.60 : Horizontal Scaling of the majority overlapping area relevant to Stop Words Usage for Request Clarification Comments

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to Stop Words Usage of Comments.

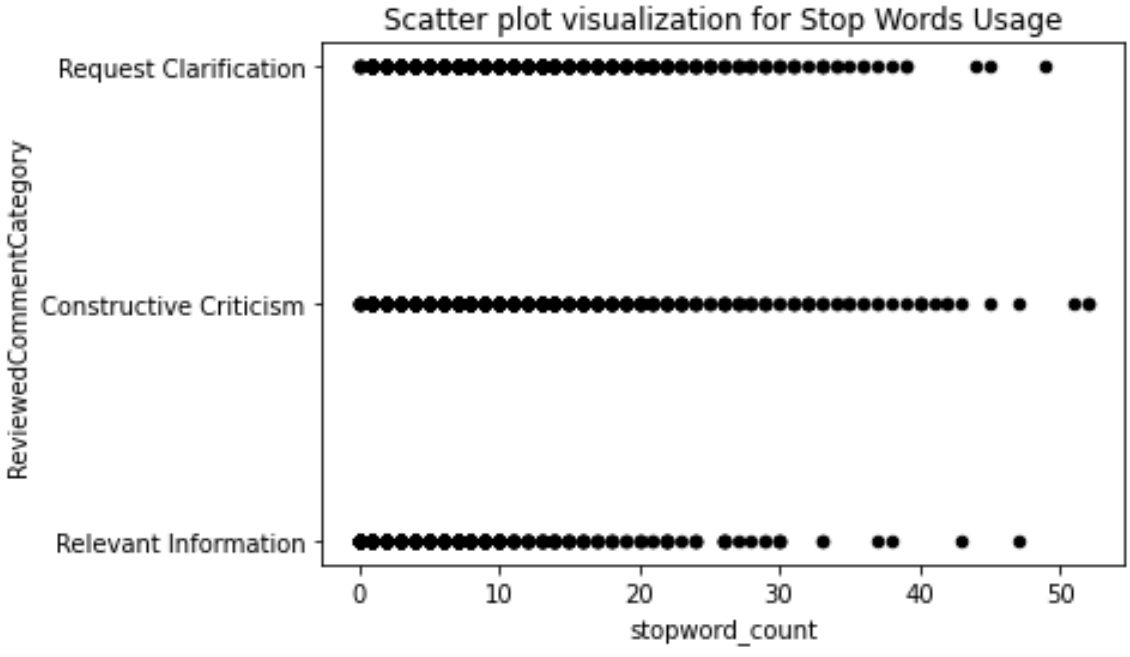


Figure 8.61 : Scatter plot visualization for Stop Words Usage

Since there is a majority overlapping area between 3 Comment Categories, the final decision was to disregard the Stop Words Usage of Comments as a feature during the implementation of the Classification Model.

8.3.7. Feature Engineering for Sentiment Score

NLTK's VADER which is a parsimonious rule based model and was used in calculating the Sentiment Score of the Comments data. VADER calculates the sentiment score of a text in terms of positive sentiment score, negative sentiment score, neutral sentiment score and normalized compound score. The compound score is the sum of negative sentiment score, positive sentiment score & neutral sentiment score which is then normalized and ranges between -1 and +1.

The following code segments contain the logic of feature creation, feature evaluation and scatter plot visualization of positive sentiment score, negative sentiment score, neutral sentiment score and normalized compound score for the comments of all 3 comment categories.

```
# Feature Engineering Trial related to Sentiment Score of Comments

nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()

df['neg'] = df['Text'].apply(lambda x:analyzer.polarity_scores(x)['neg'])
df['neu'] = df['Text'].apply(lambda x:analyzer.polarity_scores(x)['neu'])
df['pos'] = df['Text'].apply(lambda x:analyzer.polarity_scores(x)['pos'])
df['compound'] = df['Text'].apply(lambda x:analyzer.polarity_scores(x)['compound'])

bins = np.linspace(0,0.8,50)
pyplot.title("Plot related to Feature Evaluation of Score of Positive Sentiment for the entire
pyplot.xlabel('Score of Positive Sentiment')
pyplot.ylabel('Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['pos'], bins, label
            = 'Relevant Information', color = 'darkorange')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['pos'], bins, label
            = 'Request Clarification', color = 'burlywood')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['pos'], bins, label
            = 'Constructive Criticism', color = 'olivedrab')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score
pyplot.show()
```

Figure 8.62 : Feature Creation for all sentiment scores and Feature Evaluation Related to Positive Sentiment Score

```
bins = np.linspace(0,0.08,50)
pyplot.title("Plot related to Feature Evaluation of Score of Negative Sentiment for the entire Comments data")
pyplot.xlabel('Score of Negative Sentiment')
pyplot.ylabel('Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['neg'], bins, label =
            'Relevant Information', color = 'c')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['neg'], bins, label =
            'Request Clarification', color = 'b')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['neg'], bins, label =
            'Constructive Criticism', color = 'm')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Neg_Entire_SS.png')
pyplot.show()

bins = np.linspace(0.2,1,50)
pyplot.title("Plot related to Feature Evaluation of Score of Neutral Sentiment for the entire Comments data")
pyplot.xlabel('Score of Neutral Sentiment')
pyplot.ylabel('Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['neu'], bins, label =
            'Relevant Information', color = 'gold')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['neu'], bins, label =
            'Request Clarification', color = 'silver')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['neu'], bins, label =
            'Constructive Criticism', color = 'navy')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Neu_Entire_SS.png')
pyplot.show()
```

Figure 8.63: Feature Evaluation Related to Negative and Neutral Sentiment Scores

```
bins = np.linspace(-1,1,50)
pyplot.title("Plot related to Feature Evaluation of Normalized Compound Score for the entire Comments data")
pyplot.xlabel('Normalized Compound Score')
pyplot.ylabel('Number of Comments')
pyplot.hist(df[df['ReviewedCommentCategory']=='Relevant Information']['compound'], bins, label =
            'Relevant Information', color = 'turquoise')
pyplot.hist(df[df['ReviewedCommentCategory']=='Request Clarification']['compound'], bins, label =
            'Request Clarification', color = 'slateblue')
pyplot.hist(df[df['ReviewedCommentCategory']=='Constructive Criticism']['compound'], bins, label =
            'Constructive Criticism', color = 'hotpink')
pyplot.legend(loc='upper right')
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Compound_Entire_SS.png')
pyplot.show()
```

Figure 8.64 : Feature Evaluation Related to Normalized Compound Score

```
df.plot.scatter(x='pos',
                      y='ReviewedCommentCategory',
                      c='Red')
pyplot.title("Scatter plot visualization for the Score of Positive Sentiment")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Pos_Scatter_SS.png')

df.plot.scatter(x='neg',
                      y='ReviewedCommentCategory',
                      c='Blue')
pyplot.title("Scatter plot visualization for the Score of Negative Sentiment")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Neg_Scatter_SS.png')
```

Figure 8.65 : Logic of Scatter Plot Visualization for Positive and Negative Sentiment Scores

```
df.plot.scatter(x='neu',
                      y='ReviewedCommentCategory',
                      c='Green')
pyplot.title("Scatter plot visualization for the Score of Neutral Sentiment")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Neu_Scatter_SS.png')

df.plot.scatter(x='compound',
                      y='ReviewedCommentCategory',
                      c='Yellow')
pyplot.title("Scatter plot visualization for the Normalized Compound Score")
pyplot.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Sentiment_Score/Compound_Scatter_SS.png'
```

Figure 8.66 : Logic of Scatter Plot Visualization for Neutral and Compound Scores

The plot related to Feature Evaluation of Positive Sentiment depicts that majority data of Request Clarification, Relevant Information and Constructive Criticism overlaps with respect to the Positive Sentiment Score of Comments. Moreover, it depicts that the majority of useful comments contain a positive score which is near to 0 and ranges between 0 and 0.5 .

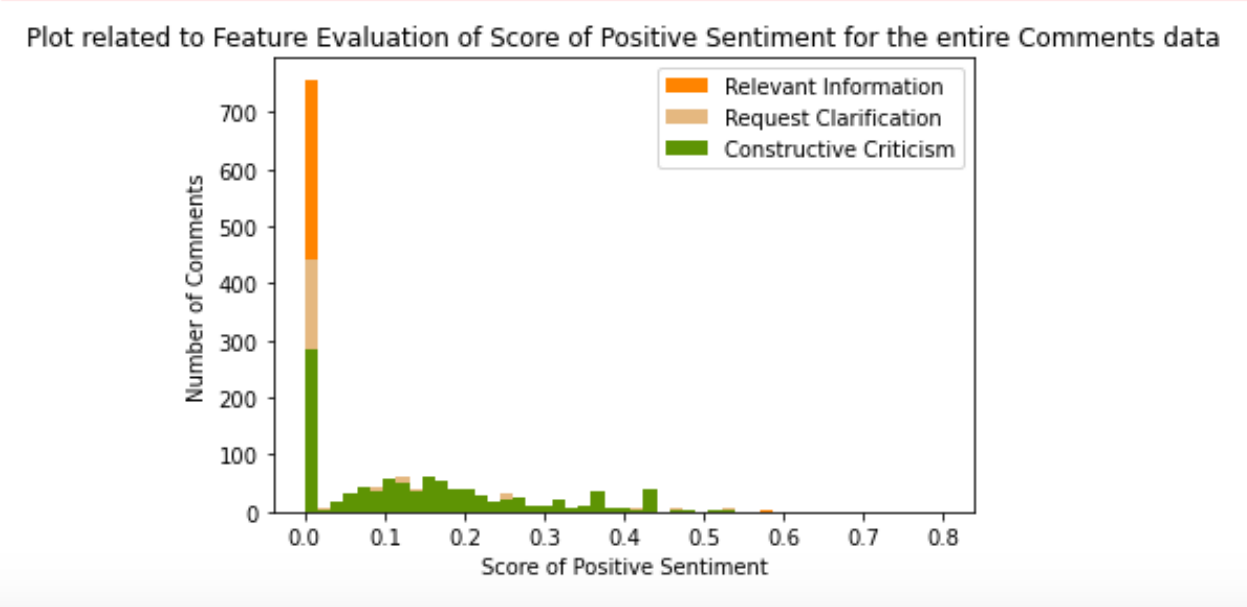


Figure 8.67 : Plot related to Feature Evaluation of Score of Positive Sentiment for the entire Comments data

The plot related to Feature Evaluation of Negative Sentiment depicts that majority data of Request Clarification, Relevant Information and Constructive Criticism overlaps with respect to the Negative Sentiment Score of Comments. Moreover, it depicts that useful comments contain a negative score which is near to 0.

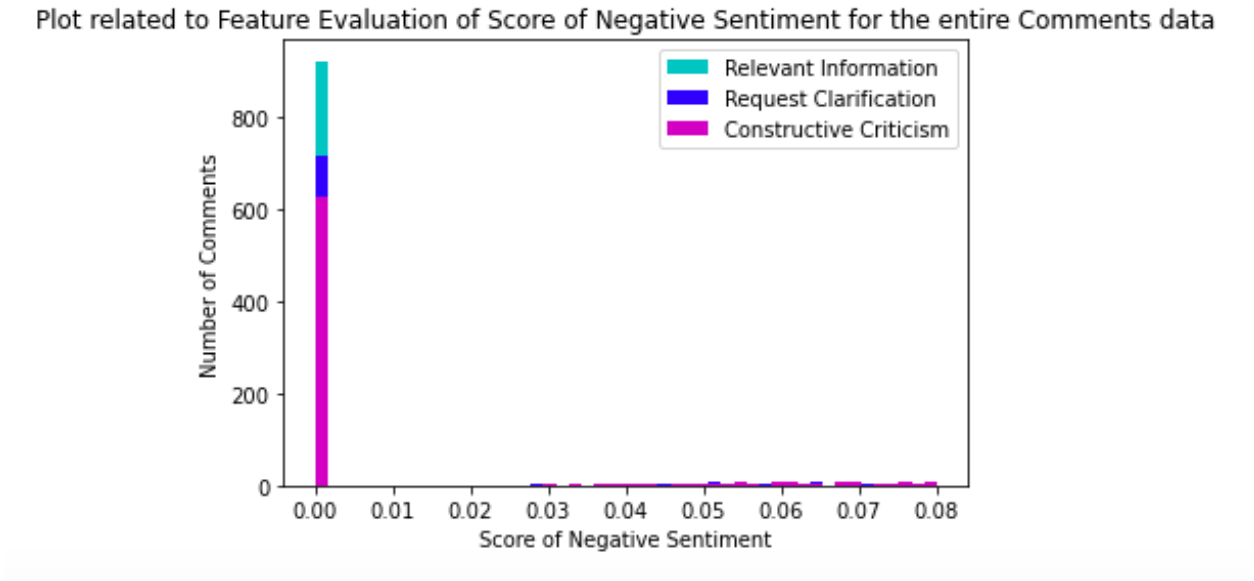


Figure 8.68 : Plot related to Feature Evaluation of Score of Negative Sentiment for the entire Comments data

The plot related to Feature Evaluation of Neutral Sentiment depicts that majority data of Request Clarification, Relevant Information and Constructive Criticism overlaps with respect to the Neutral Sentiment Score of Comments. Moreover, it depicts that the majority of useful comments contain a neutral score which is near to 1 and ranges between 0.5 and 1 .

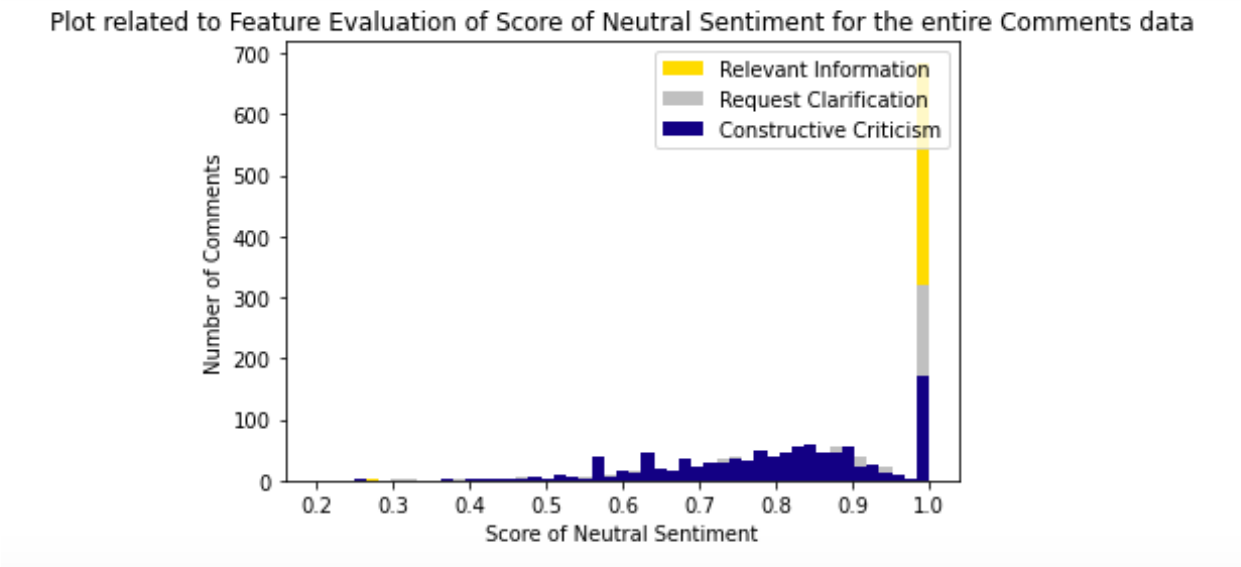


Figure 8.69 : Plot related to Feature Evaluation of Score of Neutral Sentiment for the entire Comments data

The plot related to Feature Evaluation of Normalized Compound Score depicts that majority data of Request Clarification, Relevant Information and Constructive Criticism overlaps with respect to the Normalized Compound Score of Comments. Moreover, it depicts that the majority of useful comments contain a Normalized Compound Score which is near to 0 which is the score of neutral sentiment within the range of -1 and +1.

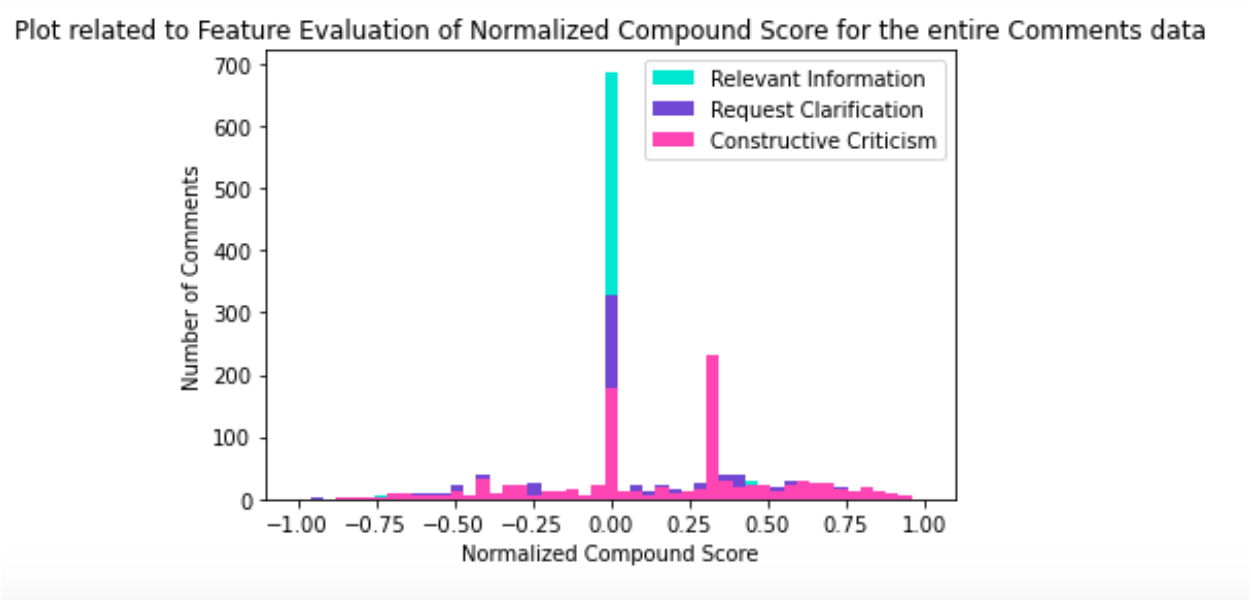


Figure 8.70 : Plot related to Feature Evaluation of Normalized Compound Score for the entire Comments data

The scatter plot distribution was drawn for the purpose of visually identifying the concentrated data distribution with respect to positive sentiment score, negative sentiment score, neutral sentiment score and normalized compound score of Comments as shown below.

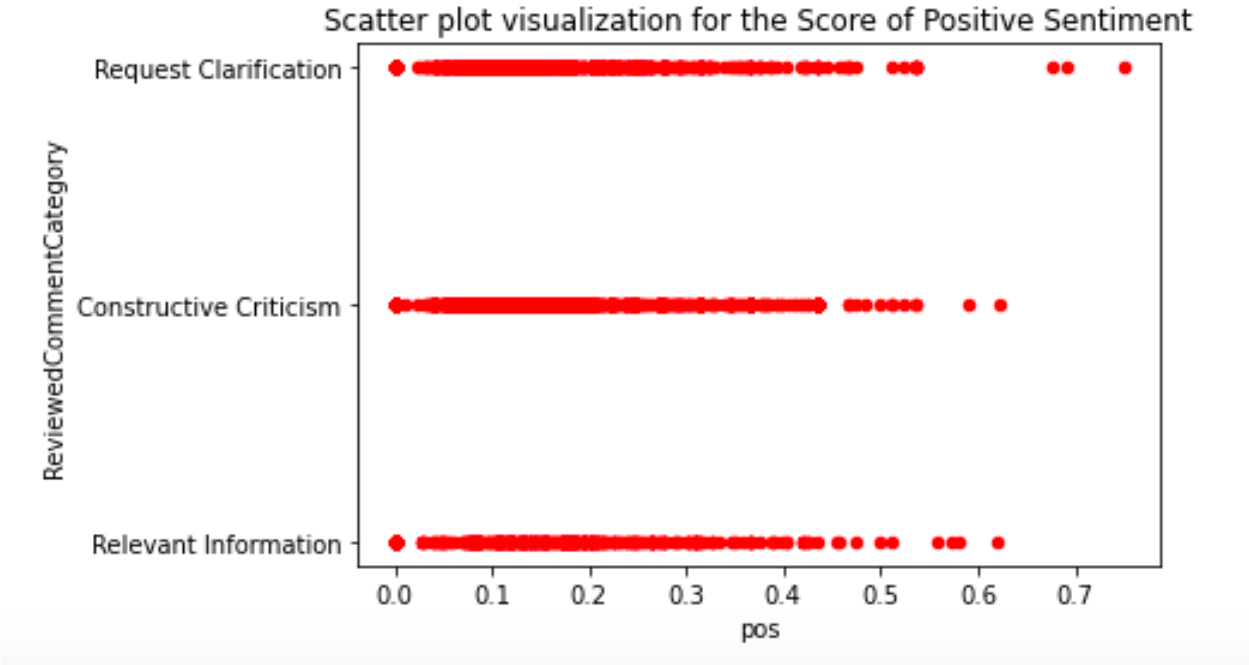


Figure 8.71 : Scatter plot visualization for the Score of Positive Sentiment

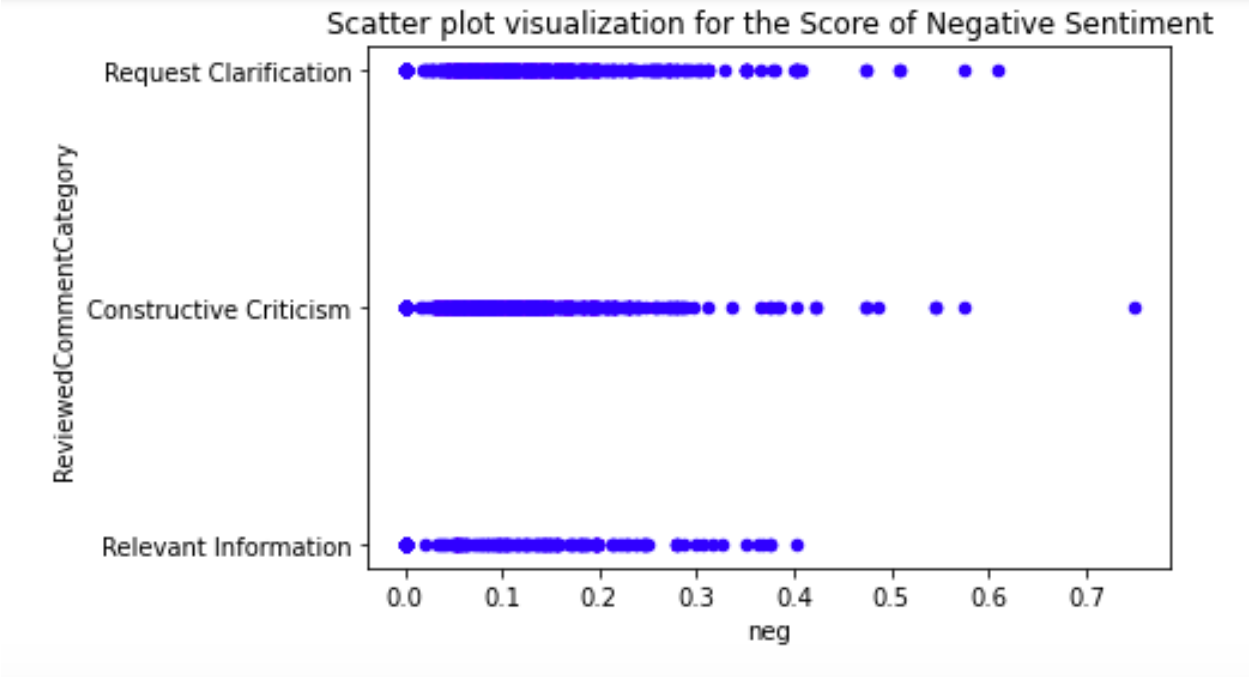


Figure 8.72 : Scatter plot visualization for the Score of Negative Sentiment

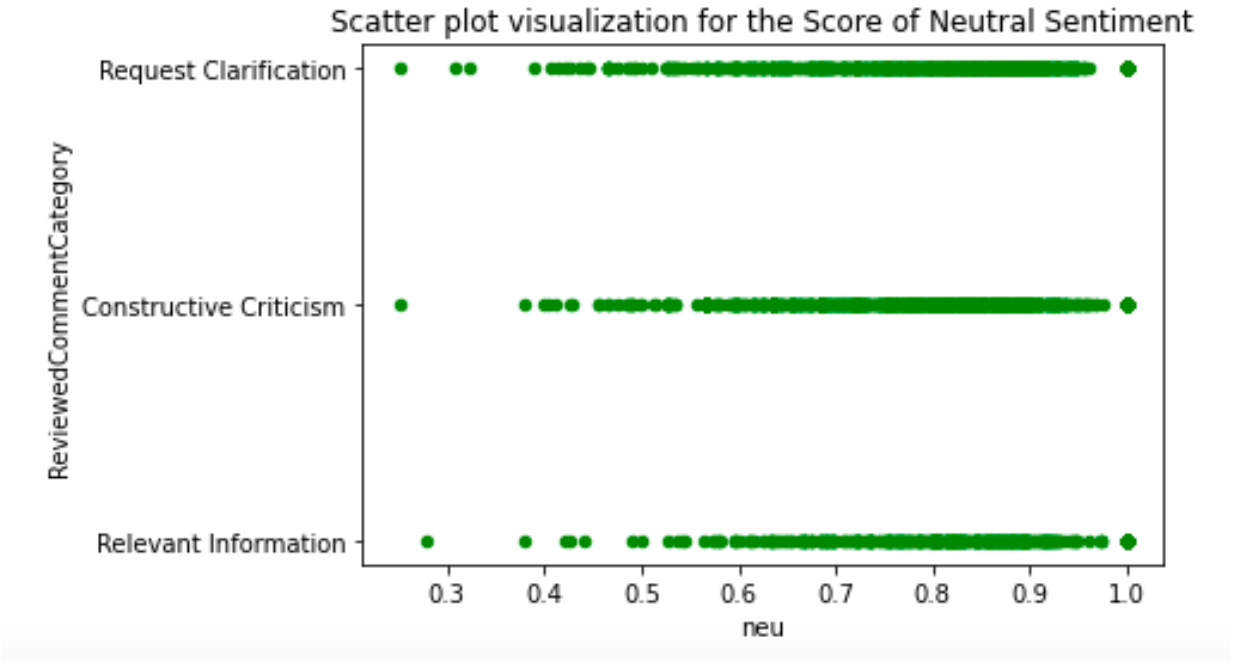


Figure 8.73: Scatter plot visualization for the Score of Neutral Sentiment

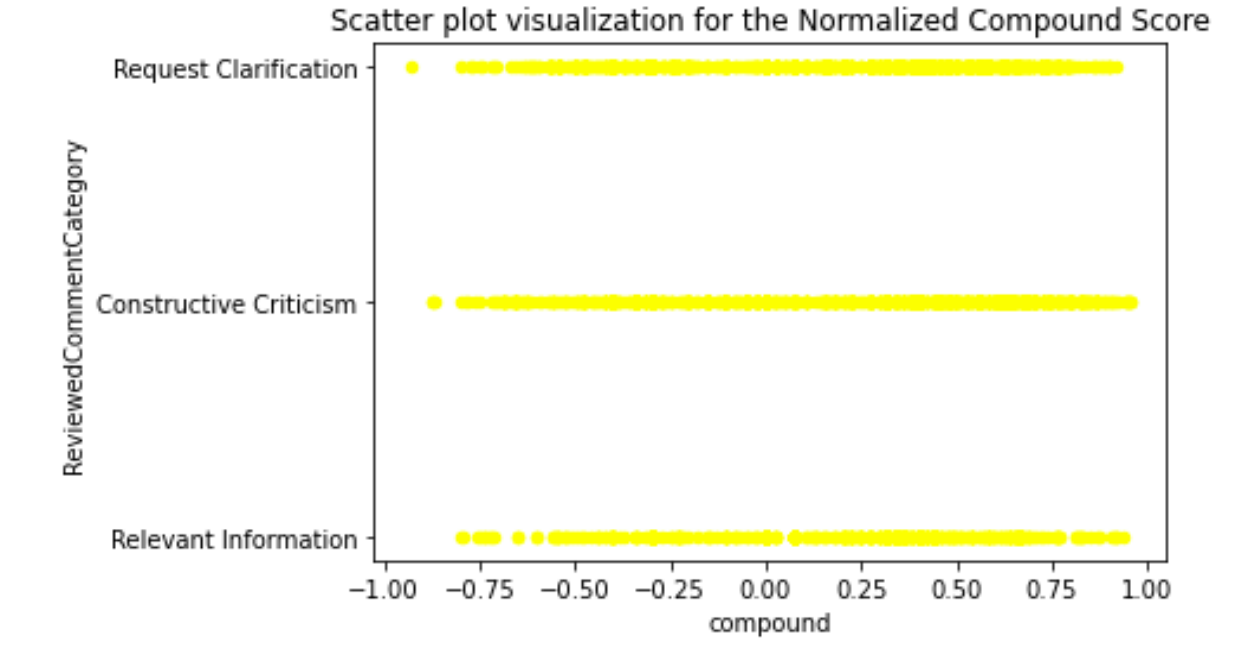


Figure 8.74 : Scatter plot visualization for the Normalized Compound Score

Since there is a majority overlapping area between all 3 Comment Categories, the final decision was to disregard the Sentiment Scores of Comments as a feature during the implementation of the Classification Model.

8.4. Preprocessing of Useful Comments

The comment text data and the Reviewed comment category labels were separated from the derived Useful comments. Then data visualisation was done for the purpose of getting an idea about the number of comments which have the character length for each range of 0-200, 200-400 and 400-600. Then the average length of comments were calculated. Preprocessing of the derived useful comments was conducted. This included lowercasing of the comment data, replacing the URLs with the keyword 'link', punctuation removal, number removal, removal of emojis and emoticons, replacement of chat words with their respective meaningful phrases, removal of stop words, tokenizing of data, Stemming and Lemmatization of the 3120 useful comment data which was identified during the qualitative analysis of the 6164 total comments.

```
#Separate the comment text data and comment category labels
comment = df['Text']
print(comment.head())
comment = comment.to_numpy()
label = df['ReviewedCommentCategory']
print(label.head())
label = label.to_numpy()
```

```
0    ngOnChanges is a lifecycle hook that fires whe...
1    For future questions: code and error messages ...
2    So you recognised the arrow function and the t...
3    `display(n->left);` but `display` doesn't expe...
4    If it is happening only in release builds, the...
Name: Text, dtype: object
0      Relevant Information
1    Constructive Criticism
2     Request Clarification
3     Request Clarification
4      Relevant Information
Name: ReviewedCommentCategory, dtype: object
```

```
x = [len(comment[i]) for i in range(comment.shape[0])]
```

```
# Calculating the Average Length of comments
print('average length of comment: {:.3f}'.format(sum(x)/len(x)) )
```

```
average length of comment: 138.029
```

Figure 8.75 : Separation of comment text data comment category labels

As in the code below, the comment length range graph was depicted to make sure that all 3120 comments lie between the 0-600 character count as more lengthy comments might be a reason for the decrease in the Accuracy of the Classification Model. Since 600 characters were taken as the comment length threshold all of the 3120 comments were taken into consideration for preprocessing.

```
bins = [1,200,400,600,800]
plt.hist(x, bins=bins)
plt.title("Plot of Comment Length Range relevant to all comments as a whole")
plt.xlabel('Length of comments (in characters)')
plt.ylabel('Number of comments')
plt.axis([0, 600, 0, 2500])
plt.grid(True)
plt.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/CommentLength_Range_for_All_Categories.png'
plt.show()
```

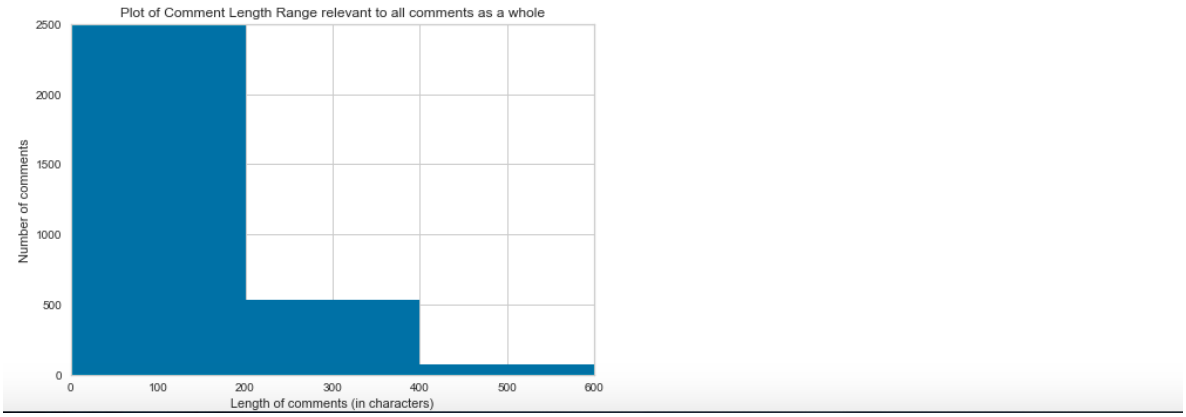


Figure 8.76 : Plotting the Comment Length Range graph for all comments

The below code snippet shows the logic related to the removal of punctuation, creation of objects for stemming and lemmatization and downloading words from the wordnet library.

```
# Whole length of comments are taken into consideration
comments = []
labels = []
for ix in range(comment.shape[0]):
        comments.append(comment[ix])
        labels.append(label[ix])
labels = np.asarray(labels)
print(len(comments))
```

```
3120
```

```
print(string.punctuation)
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

```
#Logic related to the Removal of Punctuation
translator = str.maketrans('', '', string.punctuation)
```

```
#create objects for stemmer and lemmatizer
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
```

```
#download words from wordnet library
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/prasadhiranasinghe/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Figure 8.77 : Removal of Punctuation, Creation of objects for stemmer and lemmatizer and downloading words from wordnet library

Since Comments are developer discussions, during the qualitative analysis Comments with emojis and emoticons were found. Therefore, it is necessary to handle such emojis and emoticons before feeding data for training into the Classification model.

The following code snippets contain the logic of removing emojis and emoticons from the Comments data.

```
#Logic related to the Removal of emojis
def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002500-\U00002BEF"  # chinese char
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
                               u"\u200d"
                               u"\u23cf"
                               u"\u23e9"
                               u"\u231a"
                               u"\ufe0f"  # dingbats
                               u"\u3030"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)
```

Figure 8.78 : Logic related to Removal of Emojis

```
#List of Emoticons
EMOTICONS = {
    u":-\)":"Happy face or smiley",
    u":\)":"Happy face or smiley",
    u":-\]":"Happy face or smiley",
    u":\]":"Happy face or smiley",
    u":-3":"Happy face smiley",
    u":3":"Happy face smiley",
    u":->":"Happy face smiley",
    u":>":"Happy face smiley",
    u"8-\)":"Happy face smiley",
    u":o\)":"Happy face smiley",
    u":-\}":"Happy face smiley",
    u":\}":"Happy face smiley",
    u":-\)":"Happy face smiley",
    u":c\)":"Happy face smiley",
    u":\^\)":"Happy face smiley",
    u"=\]":"Happy face smiley",
    u"=\)":"Happy face smiley",
    u":-D":"Laughing, big grin or laugh with glasses",
    u":D":"Laughing, big grin or laugh with glasses",
    u"8-D":"Laughing, big grin or laugh with glasses",
    u"8D":"Laughing, big grin or laugh with glasses",
    u"X-D":"Laughing, big grin or laugh with glasses",
    u"XD":"Laughing, big grin or laugh with glasses",
    u"=D":"Laughing, big grin or laugh with glasses",
    u"=3":"Laughing, big grin or laugh with glasses",
    u"B\^D":"Laughing, big grin or laugh with glasses",
    u":-\)\)":"Very happy",
```

Figure 8.79 : Part of the List of Emoticons which are removed from the Comments

```
#Logic related to the Removal of emoticons
def remove_emoticons(text):
    emoticon_pattern = re.compile(u'(' + u'|'.join(k for k in EMOTICONS) + u')')
    return emoticon_pattern.sub(r'', text)
```

Figure 8.80: Logic related to Removal of Emoticons

The code snippet below contains the logic of replacing the chat words with its respective meaningful phrases. For the replacement, a text document containing a list of chat words was utilized.

```
AFAIK=As Far As I Know
AFK=Away From Keyboard
AI=Artificial Intelligence
ALU=Arithmetic Logic Unit
API=Application Programming Interface
ARPANET=Advanced Research Project Agency Network
ASAP=As Soon As Possible
ASCII=American Standard Code for Information Interchange
ATK=At The Keyboard
ATM=At The Moment
A3=Anytime, Anywhere, Anyplace
BAK=Back At Keyboard
BCZ=Because
BIOS=Basic Input Output System
BRB=Be Right Back
BRT=Be Right There
BTW=By The Way
B4=Before
B4N=Bye For Now
CAD=Computer Aided Design
CAE=Computer Aided Engineering
CD=Compact Disk
CD-ROM=Compact Disk Read Only Memory
CLI=Command Line Interface
CPU=Central Processing Unit
CV=Close Vote
CVer=Close Voter
CW=Community Wiki
DBMS=Data Base Management System
DV=Delete vote
ENIAC=Electronic Numerical Integrator And Calculator
FAQ=Frequently Asked Questions
FGITW=Fastest Gun in the West
FHRC=Free Hand Red Circle
FORTRAN=FORmula TRANslation
FC=Fingers Crossed
FTP=File Transfer Protocol
FWIW=For What It's Worth
FYI=For Your Information
FR=Feature Request
GAL=Get A Life
GB=Giga Bytes
GHz=Giga Hertz
GG=Good Game
GMTA=Great Minds Think Alike
GPRS=General Packet Radio Service
GR8=Great!
GSM=Global System for Mobile communication
GUI=Graphical User Interface
HNQ=Hot Network Questions
HTML=HyperText Markup Language
IMHO=In My Honest/Humble Opinion
IKR=I Know Right
IMO=In My Opinion
IOW=In Other Words
IRL=In Real Life
JPEG=Joint Photographic Experts Group
JRE=Java Runtime Environment
KB=Kilo Bytes
```

Figure 8.81 : Part of the list of utilized chat words and their respective meanings

```
#Logic related to handling of chatwords in comments
def chatword_handler(text):
    text = text.split(" ")
    j = 0
    for _str in text:
        # File path which consists of Abbreviations.
        fileName = "ChatWords.txt"
        # File Access mode [Read Mode]
        accessMode = "r"
        with open(fileName, accessMode) as myCSVfile:
            # Reading file as CSV with delimiter as "=", so that abbreviation are stored in row[0] and phrases in row[1
            dataFromFile = csv.reader(myCSVfile, delimiter="=")
            # Removing Special Characters.
            _str = re.sub('[^a-zA-Z0-9-_.]', '', _str)
            for row in dataFromFile:
                # Check if selected word matches short forms[LHS] in text file.
                if _str.upper() == row[0]:
                    # If match found replace it with its appropriate phrase in text file.
                    text[j] = row[1]
            myCSVfile.close()
        j = j + 1
    # Replacing commas with spaces for final output.
    return(' '.join(text))
```

Figure 8.82 : Logic related to the handling of chat words

As the Relevant Information Comments captures links to related question & answers and Informative Links redirecting to other websites, it was necessary to implement a logic of replacing such links with a meaningful keyword. Therefore such links were replaced with the key word ‘link’. The logic below contains replacing URLs with the keyword ‘link’.

```
#Logic related to replacing URLs with String
def replace_urls(text):
    url_replaced_comment = re.sub(r'https?://\S+|www\.\S+', "link", text)
    return url_replaced_comment
```

Figure 8.83: Logic related to replacing URLs with the keyword ‘link’

Then the application of all above mentioned preprocessing steps was done for all 3120 comments.

```
for i in range(len(comments)):
    List_Without_stopWords = []

    #Lowercasing comments
    comments[i] = comments[i].lower()

    #Replacing URL with a keyword in comments
    comments[i] = replace_urls(comments[i])

    #Removing Numbers from comments
    comments[i] = re.sub(r'\d+', '', comments[i])

    #Removing Punctuation from comments
    comments[i] = comments[i].translate(translator)

    #Removing emojis from comments
    comments[i] = remove_emoji(comments[i])

    #Removing emoticons from comments
    comments[i] = remove_emoticons(comments[i])

    #Replacing chat words in comments
    comments[i] = chatword_handler(comments[i])

    nltk_tokens = nltk.word_tokenize(comments[i])
    #print(nltk_tokens)

    for word in nltk_tokens:
        if word not in stopwords:
            List_Without_stopWords.append(word)
```

Figure 8.84 : Lowercasing, Replacement of URLs with a keyword, Removal of Punctuation, numbers, emojis, emoticons and Stop words, Tokenization and Replacement of Chatwords of comments

```
# Applying Stemmer and Lemmatizer
l = []
for word in List_Without_stopWords:
    l.append(stemmer.stem(lemmatizer.lemmatize(word,pos="v")))
comments[i] = " ".join(l)

df.loc[i,'text_final'] = str(comments[i])
```

Figure 8.85 : Application of Stemming and Lemmatization for all comments data

Then the output comments in which Lowercasing, Replacement of URLs with a keyword, Removal of Punctuation, numbers, emojis, emoticons and Stop words, Tokenization, Replacement of Chat Words, Stemming and Lemmatization have been performed is saved in "Preprocessed_Comments.csv" file as shown below.

```
print(df['text_final'].head())
```

```
0    ngonchang lifecycl hook fire input properti ch...
1    futur question code error messag question post...
2    stack overflow recognis arrow function templat...
3    displaynleft display doesnt expect paramet mea...
4    happen releas build proguard issu check quick ...
Name: text_final, dtype: object
```

```
for i in range(len(labels)):
    df.loc[i, 'label'] = str(labels[i])
```

```
with open('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Output_ResultingDatasets/Preprocessed_Comments.csv
    writer = csv.writer(file)
    for i in range(len(comments)):
        writer.writerow([comments[i]])
```

Figure 8.86 : Writing the Preprocessed Comments data to a separate CSV file

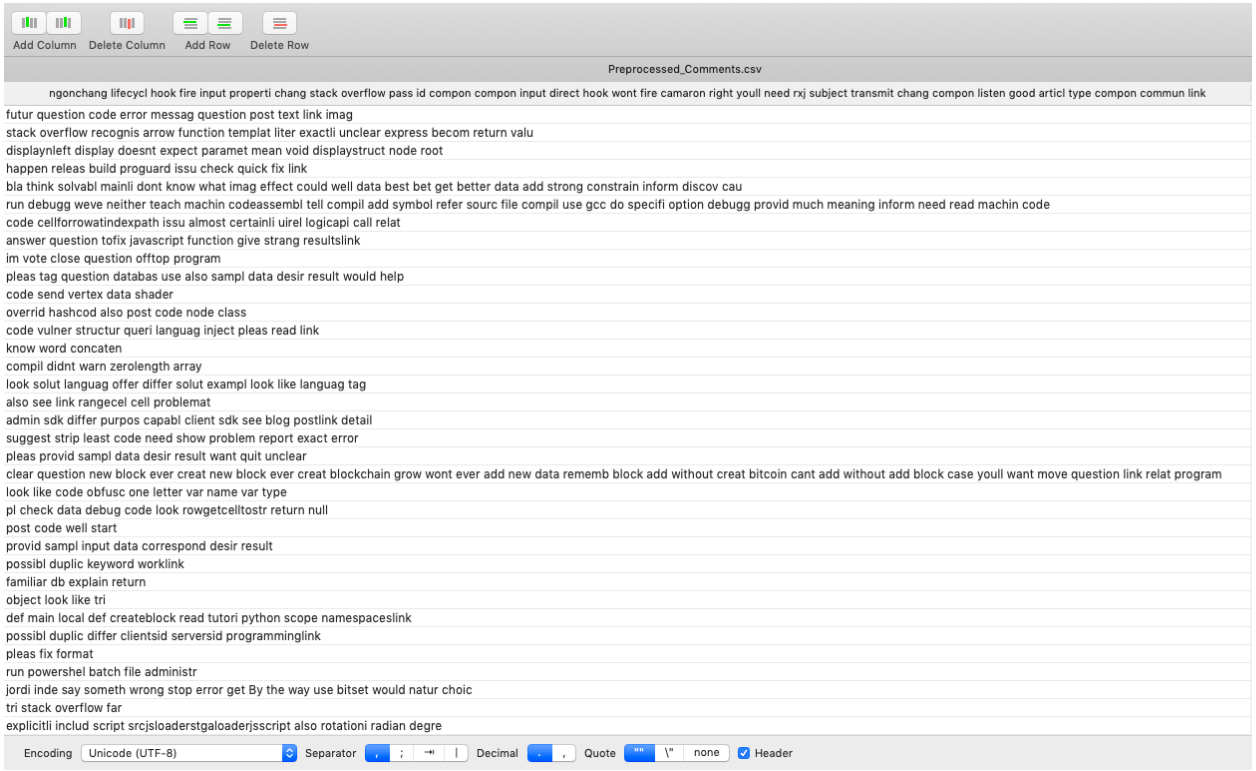


Figure 8.87: Useful Comments after Lowercasing, Replacement of URLs with a keyword,

Removal of Punctuation, numbers, emojis, emoticons and Stop words, Tokenization, Replacement of Chat Words, Stemming and Lemmatization

8.5. Feature Extraction of Comments data

After the preprocessing stage, feature extraction using TF-IDF and N-grams was performed.
The following code snippet contains the logic of importing necessary libraries for Feature Extraction and summarizing the number of comments in each comment category.

```
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsOneClassifier
```

```
features = df[['text_final','label']]
features.head()
```

	text_final	label
0	ngonchang lifecycl hook fire input properti ch...	Relevant Information
1	futur question code error messag question post...	Constructive Criticism
2	stack overflow recognis arrow function templat...	Request Clarification
3	displaynleft display doesnt expect paramet mea...	Request Clarification
4	happen releas build proguard issu check quick ...	Relevant Information

```
features.label.value_counts()
```

```
Request Clarification     1063
Relevant Information      1047
Constructive Criticism    1010
Name: label, dtype: int64
```

Figure 8.88 : Importing necessary libraries for Feature Extraction and summarizing the number of comments in each comment category

Encoding of the categorical labels were necessary as the labels are fed into the training model with numeric values. Hence, the encoding was done using the LabelEncoder. Afterwards, the train-test split was done with 80% of data for training and 20% of data for testing. Shuffling of data was performed to avoid biases of data location to be fed into the training model. Random

state was set in order to decide the splitting of train-test indices. Afterwards training and testing data were written into separate CSV files.

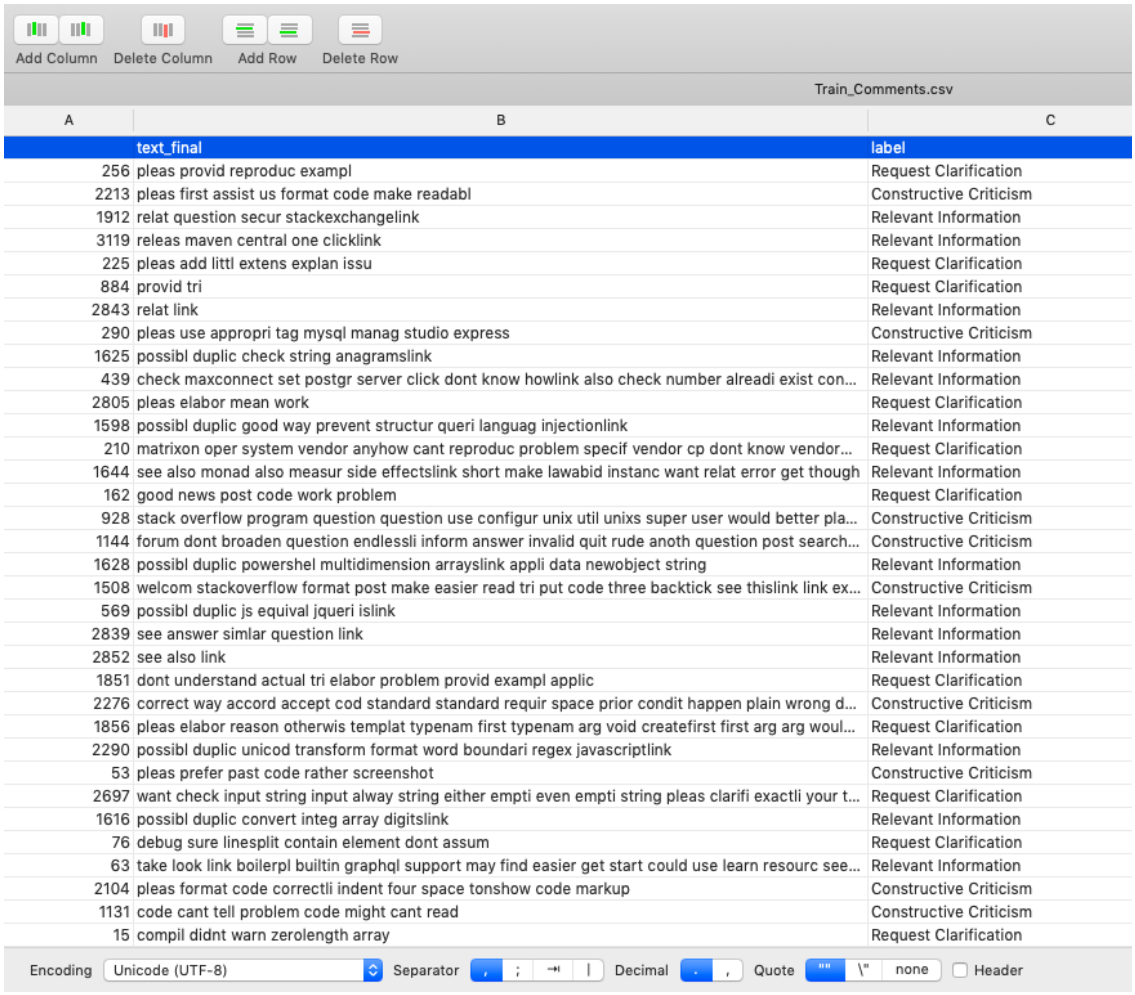
```
Encoder = LabelEncoder()
df1 = features[['label']]
df1['Respective Encoding'] = Encoder.fit_transform(features['label'])
df1.value_counts()
```

```
label                   Respective Encoding
Request Clarification   2                     1063
Relevant Information    1                     1047
Constructive Criticism  0                     1010
dtype: int64
```

```
# Train-Test split with 80% for training set and 20% for testing set while shuffling is performed
Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(df['text_final'], df['label'],
                                                                    train_size=0.80, test_size=0.2,
                                                                    shuffle=True, random_state=1001)
```

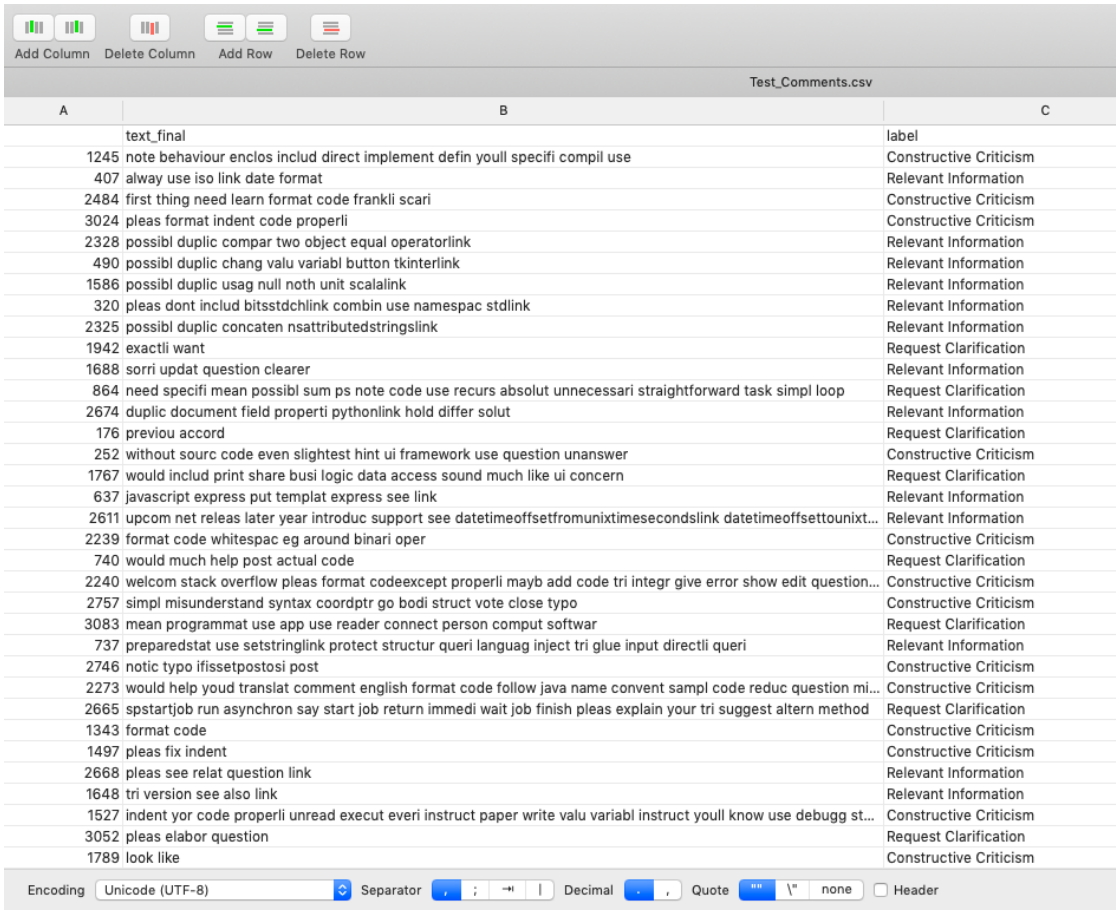
```
# Writing the Training and Testing data to separate files after shuffling

Test_data = pd.concat([Test_X, Test_Y], axis=1)
Train_data = pd.concat([Train_X, Train_Y], axis=1)

Test_data.to_csv('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Output_ResultingDatasets/Test_Comments.csv')
Train_data.to_csv('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Output_ResultingDatasets/Train_Comments.csv
```

Figure 8.89 : Getting the respective encoding for each comment label, Performing Train-Test Split, and writing the training data and testing data in separate CSV files



A	B	C
	text_final	label
256	pleas provid reproduc exampl	Request Clarification
2213	pleas first assist us format code make readabl	Constructive Criticism
1912	relat question secur stackexchangelink	Relevant Information
3119	releas maven central one clicklink	Relevant Information
225	pleas add littl extens explan issu	Request Clarification
884	provid tri	Request Clarification
2843	relat link	Relevant Information
290	pleas use appropri tag mysql manag studio express	Constructive Criticism
1625	possibl duplic check string anagramslink	Relevant Information
439	check maxconnect set postgr server click dont know howlink also check number alreadi exist con...	Relevant Information
2805	pleas elabor mean work	Request Clarification
1598	possibl duplic good way prevent structur queri languag injectionlink	Relevant Information
210	matrixon oper system vendor anyhow cant reproduc problem specif vendor cp dont know vendor...	Request Clarification
1644	see also monad also measur side effectslink short make lawabid instanc want relat error get though	Relevant Information
162	good news post code work problem	Request Clarification
928	stack overflow program question question use configur unix util unixs super user would better pla...	Constructive Criticism
1144	forum dont broaden question endlessli inform answer invalid quit rude anoth question post search...	Constructive Criticism
1628	possibl duplic powershel multidimension arrayslink appli data newobject string	Relevant Information
1508	welcom stackoverflow format post make easier read tri put code three backtick see thislink link ex...	Constructive Criticism
569	possibl duplic js equival jqueri islink	Relevant Information
2839	see answer simlar question link	Relevant Information
2852	see also link	Relevant Information
1851	dont understand actual tri elabor problem provid exampl applic	Request Clarification
2276	correct way accord accept cod standard standard requir space prior condit happen plain wrong d...	Constructive Criticism
1856	pleas elabor reason otherwis templat typenam first typenam arg void createfirst first arg arg woul...	Request Clarification
2290	possibl duplic unicod transform format word boundari regex javascriptlink	Relevant Information
53	pleas prefer past code rather screenshot	Constructive Criticism
2697	want check input string input alway string either empti even empti string pleas clarifi exactli your t...	Request Clarification
1616	possibl duplic convert integ array digitslink	Relevant Information
76	debug sure linesplit contain element dont assum	Request Clarification
63	take look link boilerpl builtin graphql support may find easier get start could use learn resourc see...	Relevant Information
2104	pleas format code correctli indent four space tonshow code markup	Constructive Criticism
1131	code cant tell problem code might cant read	Constructive Criticism
15	compil didnt warn zerolength array	Request Clarification

Figure 8.90 : Training data after split



A	B	C
	text_final	label
1245	note behaviour enclos includ direct implement defin youll specifi compil use	Constructive Criticism
407	alway use iso link date format	Relevant Information
2484	first thing need learn format code frankli scari	Constructive Criticism
3024	pleas format indent code properli	Constructive Criticism
2328	possibl duplic compar two object equal operatorlink	Relevant Information
490	possibl duplic chang valu variabl button tkinterlink	Relevant Information
1586	possibl duplic usag null noth unit scalalink	Relevant Information
320	pleas dont includ bitsstdchlink combin use namespac stdlink	Relevant Information
2325	possibl duplic concaten nsattributedstringslink	Relevant Information
1942	exactli want	Request Clarification
1688	sorri updat question clearer	Relevant Information
864	need specifi mean possibl sum ps note code use recurs absolut unnecessari straightforward task simpl loop	Request Clarification
2674	duplic document field properti pythonlink hold differ solut	Relevant Information
176	previou accord	Request Clarification
252	without sourc code even slightest hint ui framework use question unanswer	Constructive Criticism
1767	would includ print share busi logic data access sound much like ui concern	Request Clarification
637	javascript express put templat express see link	Relevant Information
2611	upcom net releas later year introduc support see datetimeoffsetfromunixtimesecondslink datetimeoffsettounixt...	Relevant Information
2239	format code whitespac eg around binari oper	Constructive Criticism
740	would much help post actual code	Request Clarification
2240	welcom stack overflow pleas format codeexcept properli mayb add code tri integr give error show edit question...	Constructive Criticism
2757	simpl misunderstand syntax coordptr go bodi struct vote close typo	Constructive Criticism
3083	mean programmat use app use reader connect person comput softwar	Request Clarification
737	preparedstat use setstringlink protect structur queri languag inject tri glue input directli queri	Relevant Information
2746	notic typo ifissetpostosi post	Constructive Criticism
2273	would help youd translat comment english format code follow java name convent sampl code reduc question mi...	Constructive Criticism
2665	spstartjob run asynchron say start job return immedi wait job finish pleas explain your tri suggest altern method	Request Clarification
1343	format code	Constructive Criticism
1497	pleas fix indent	Constructive Criticism
2668	pleas see relat question link	Relevant Information
1648	tri version see also link	Relevant Information
1527	indent yor code properli unread execut everi instruct paper write valu variabl instruct youll know use debugg st...	Constructive Criticism
3052	pleas elabor question	Request Clarification
1789	look like	Constructive Criticism

Figure 8.91 : Testing data after split

Afterwards, the feature extraction of comments using TF-IDF and N-grams was done. Unigrams and Bigrams were used as N-grams. Min_df was set to 20 for the purpose of removing more infrequent words and to increase the Classification accuracy. English stop_words were taken into consideration. The resulting Train_X and Test_X features were written to separate files. Note that the outputs were sparse matrices.

```
# Feature Extraction process using TFIDF and N-grams features
Train_Y = Encoder.fit_transform(Train_Y.astype(str))
Test_Y =  Encoder.fit_transform(Test_Y.astype(str))

tfidf_vect_ngram = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}',
                                   ngram_range=(1,2), min_df=20, stop_words='english', max_features=None)
fitted_vectorizer=tfidf_vect_ngram.fit(df['text_final'].astype(str))
Train_X=fitted_vectorizer.transform(Train_X.astype(str))
Test_X = fitted_vectorizer.transform(Test_X.astype(str))
```

```
# Writing extracted features on separate files

with open('/Users/prasadhiranasinghe/Desktop/Output_FeatureFiles_FYP/TrainX_Features.csv', 'w', newline='') as file:
    mywriter = csv.writer(file, delimiter=',')
    mywriter.writerows(Train_X.toarray())

with open('/Users/prasadhiranasinghe/Desktop/Output_FeatureFiles_FYP/TestX_Features.csv', 'w', newline='') as file:
    mywriter = csv.writer(file, delimiter=',')
    mywriter.writerows(Test_X.toarray())
```

Figure 8.92 : Feature Extraction using TF-IDF and N-grams and writing the output of extracted features in separate CSV files

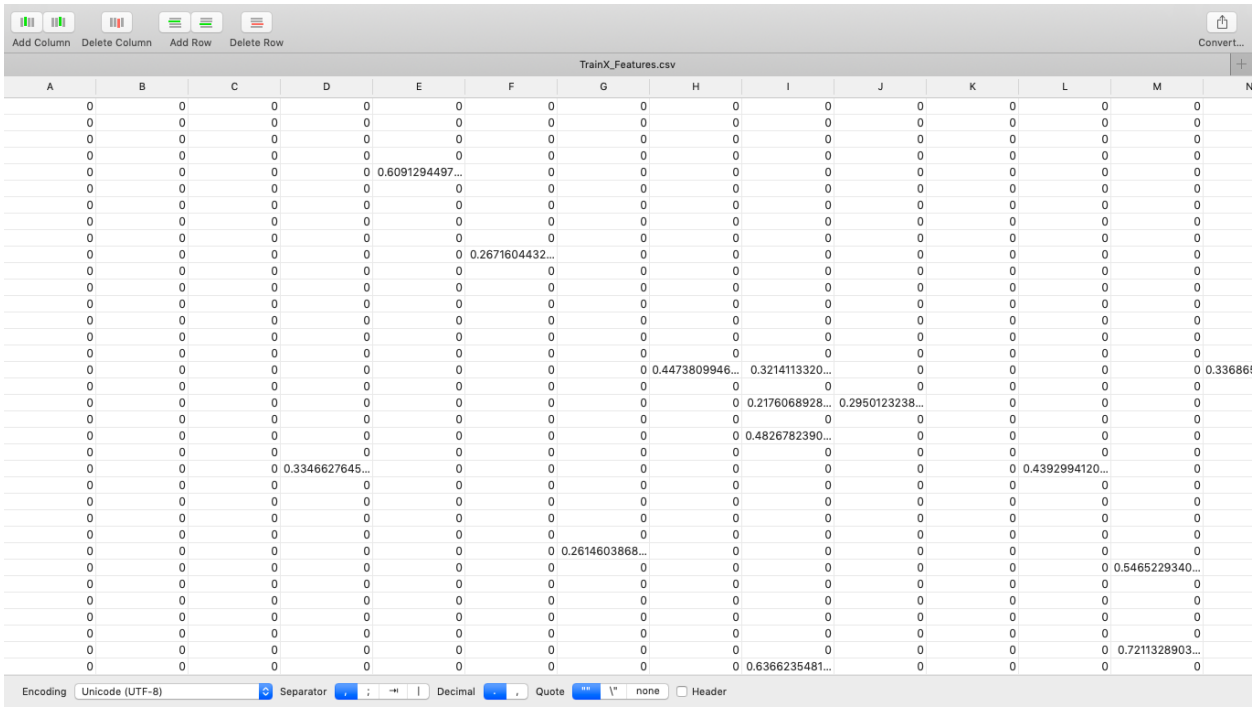


Figure 8.93 - Train_X features

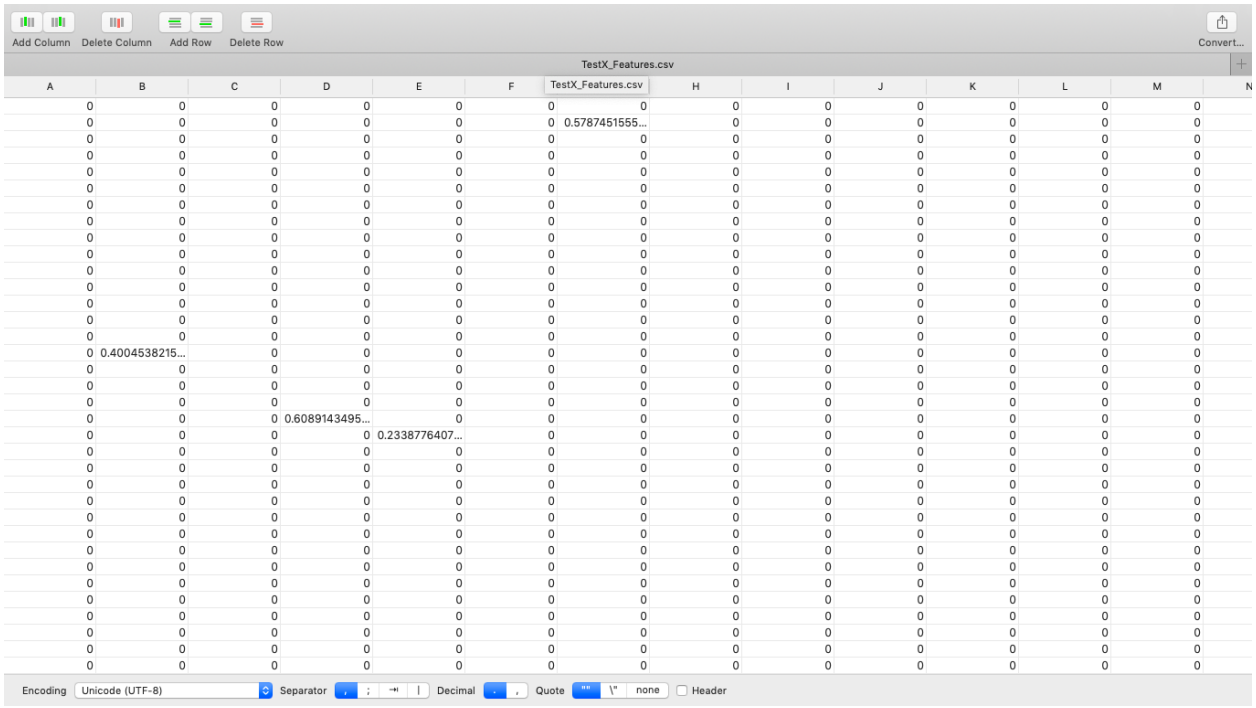


Figure 8.94 : Test_X features

The following code snippet outputs the vocabulary that the word vectorizer has learnt from the corpus.

```
# Prints the vocabulary that the word vectorizer has learnt from the corpus
print(tfidf_vect_ngram.vocabulary_)
```

```
{'input': 103, 'chang': 23, 'stack': 225, 'overflow': 146, 'right': 203, 'youll': 273, 'need': 137, 'good': 90, 'type
': 250, 'link': 117, 'stack overflow': 226, 'question': 181, 'code': 32, 'error': 68, 'messag': 131, 'post': 168, 'te
xt': 243, 'imag': 97, 'error messag': 69, 'function': 87, 'exactli': 71, 'unclear': 252, 'return': 202, 'valu': 259,
'doesnt': 58, 'expect': 74, 'paramet': 148, 'mean': 128, 'happen': 91, 'issu': 107, 'check': 25, 'fix': 81, 'think':
246, 'dont': 59, 'know': 112, 'data': 50, 'better': 17, 'add': 4, 'inform': 102, 'run': 204, 'tell': 241, 'compil': 3
9, 'refer': 190, 'file': 80, 'use': 257, 'provid': 177, 'read': 186, 'relat': 193, 'answer': 8, 'javascript': 110, 'a
nswer question': 9, 'im': 96, 'vote': 264, 'close': 31, 'program': 174, 'pleas': 154, 'tag': 240, 'sampl': 205, 'desi
r': 54, 'result': 201, 'help': 93, 'sampl data': 206, 'class': 29, 'post code': 169, 'structur': 232, 'queri': 179, '
languag': 113, 'structur queri': 233, 'queri languag': 180, 'pleas read': 164, 'word': 269, 'array': 12, 'look': 120,
'solut': 218, 'differ': 55, 'exampl': 72, 'like': 115, 'look like': 121, 'suggest': 234, 'problem': 172, 'exact': 70,
'want': 265, 'pleas provid': 163, 'clear': 30, 'new': 138, 'block': 19, 'creat': 48, 'case': 22, 'question link': 182
, 'debug': 52, 'start': 228, 'possibl': 166, 'duplic': 60, 'possibl duplic': 167, 'explain': 76, 'object': 142, 'tri'
: 249, 'python': 178, 'format': 85, 'pleas fix': 160, 'say': 207, 'someth': 220, 'wrong': 272, 'way': 266, 'far': 78,
'includ': 100, 'script': 209, 'welcom': 267, 'note': 139, 'write': 271, 'effort': 64, 'make': 123, 'ask': 13, 'work':
270, 'welcom stack': 268, 'overflow pleas': 147, 'sure': 237, 'minim': 133, 'complet': 40, 'verifi': 261, 'minim comp
let': 134, 'complet verifi': 41, 'verifi exampl': 262, 'someon': 219, 'attempt': 15, 'youv': 274, 'isnt': 106, 'outpu
t': 145, 'sinc': 215, 'oper': 143, 'store': 230, 'tabl': 239, 'difficult': 56, 'format code': 86, 'actual': 3, 'line'
: 116, 'contain': 43, 'string': 231, 'method': 132, 'past': 150, 'screenshot': 208, 'sens': 211, 'anoth': 7, 'mani':
124, 'pars': 149, 'support': 235, 'easier': 61, 'learn': 114, 'resourc': 200, 'instead': 105, 'loop': 122, 'list': 11
9, 'sound': 221, 'execut': 73, 'set': 212, 'base': 16, 'alway': 6, 'correct': 46, 'probabl': 171, 'describ': 53, 'upd
at': 255, 'bite': 18, 'element': 66, 'convert': 44, 'variabl': 260, 'fail': 77, 'memori': 129, 'document': 57, 'place
': 153, 'suppos': 236, 'version': 263, 'share': 213, 'implement': 98, 'user': 258, 'reproduc': 197, 'noth': 140, 'que
stion pleas': 183, 'pleas explain': 159, 'specif': 223, 'access': 1, 'field': 79, 'reason': 189, 'date': 51, 'hyperte
xt': 94, 'markup': 125, 'hypertext markup': 95, 'markup languag': 126, 'copi': 45, 'button': 20, 'column': 37, 'git':
89, 'current': 49, 'follow': 83, 'instal': 104, 'json': 111, 'php': 152, 'number': 141, 'request': 198, 'miss': 135,
'order': 144, 'titl': 248, 'gener': 88, 'understand': 253, 'abl': 0, 'alreadi': 5, 'thank': 244, 'ive': 108, 'expect
```

Figure 8.95 : Printing the vocabulary that the word vectorizer has learnt from the corpus

The following code snippet outputs the vectorized training data.

```
# Print the vectorized training data
print(Train_X)
```

```
  (0, 197)      0.5501719257234367
  (0, 177)      0.43833960746230893
  (0, 163)      0.5501719257234367
  (0, 154)      0.22137217686659894
  (0, 72)       0.391758155155273
  (1, 187)      0.5893709077225057
  (1, 154)      0.2774072086640442
  (1, 123)      0.47964356814378756
  (1, 86)       0.3758597823873536
  (1, 85)       0.3505633077409018
  (1, 32)       0.28541901215468785
  (2, 194)      0.7182152565020591
  (2, 193)      0.5872813194492676
  (2, 181)      0.373185606841433
  (4, 154)      0.3223628693140159
  (4, 107)      0.7245988503320062
  (4, 4)        0.6091294497766213
  (5, 249)      0.6293470709369442
  (5, 177)      0.7771243557520823
  (6, 193)      0.8155608147342064
  (6, 117)      0.5786713726028595
  (7, 257)      0.4660886309063965
  (7, 240)      0.8170285856813461
  (7, 154)      0.3394490805987555
  (8, 231)      0.5425771190973517
  :     :
  (2494, 174)   0.27002851225140745
```

Figure 8.96 : Printing the vectorized training data

8.6. Building the SVM Classification Model and training the Model

Building the SVM Classification Model was complex since after the initial build of the model, hyperparameter tuning was necessary to identify the best parameters for each SVM Kernel. It was also necessary to identify the kernel which best fits the data and performs well.

8.6.1. Building the Initial SVM Classification Model

The initial SVM model was built without any parameters. The kernel was not specified during this build. As the kernel was not specified in this build of the model, the RBF Kernel will be used as the default kernel according to sklearn.

```
from sklearn.model_selection import GridSearchCV
from sklearn import svm
import pickle
```

```
# Initial SVM Model Creation

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

SVM_Model = svm.SVC().fit(Train_X, Train_Y)

# save the model
filename = 'initial_svm_model.sav'
pickle.dump(SVM_Model, open(filename, 'wb'))

SVM_Model_pred = SVM_Model.predict(Test_X)
```

Figure 8.97 : Initial SVM Model

For the initial SVM Model an evaluation of Confusion Matrix and Classification Report was done. Accuracy and F1-Scores were gained initially by the holdout method. This is further discussed in the Evaluation Chapter.

8.6.2. Hyperparameter Tuning

Hyperparameters control the behaviour of the overall machine learning model. Therefore hyperparameter tuning is necessary. The ultimate goal is to find the optimal combination of hyperparameters of the SVM Model that minimizes the loss and maximizes the overall accuracy of the Model. Since certain hyperparameter combinations are not supported with specific kernels in SVM, hyperparameters tuning was done for 5 SVM kernels separately.

The objective was to identify the best Kernel with best hyperparameter combinations.

For the Hyperparameter tuning GridSearchCV was used. Since Cross Validation was done with GridSearchCV to obtain the best combination of hyperparameters, a validation data set is no longer needed as the cross validation divides the training data set into k number of folds , and k-1 folds are used for the training purpose and the remaining fold is utilized as the validation set.

In this hyperparameter tuning, cross validation with 3 folds was performed.

The following hyperparameters were utilized in hyperparameter tuning and then building the SVM Model with the perfect combination of hyperparameters.

- C - This is the regularization parameter. This should be positive strictly.
- kernel - Specifies the kernel to be used with the classification algorithm
- degree - This is the degree of the polynomial kernel function and is ignored by the rest of the kernels.
- gamma - This is the Kernel coefficient
- decision_function_shape - Determines whether to use One-Vs-Rest(ovr) decision function shape or One-Vs-One(ovo) decision function shape. For the multi class strategy One-Vs-One(ovo) is used often.
- coef0 - It is an independent term in the kernel function which is significant for Sigmoid and Polynomial Kernels most of the time.

8.6.2.1. Hyperparameter Tuning for Linear Kernel

The following code snippet contains the logic of hyperparameter tuning for the Linear Kernel of SVM.

```
# Hyperparameter tuning for SVM's Linear Kernel

# defining parameter range
param_grid_linear = {'C': [0.1, 1, 2, 3, 5, 10, 100, 1000],
                     'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'auto', 'scale'],
                     'decision_function_shape': ['ovo', 'ovr'],
                     'kernel': ['linear']}

grid_linear = GridSearchCV(svm.SVC(), param_grid_linear, refit = True, verbose = 2, cv=3)

# fitting the model for grid search
grid_linear.fit(Train_X,Train_Y)

# print best parameter after tuning
print("Best parameters of Linear Kernel after tuning are: ", grid_linear.best_params_)
print("\n")
# print how our model looks after hyper-parameter tuning
print(grid_linear.best_estimator_)
```

```
Fitting 3 folds for each of 112 candidates, totalling 336 fits
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=linear ......

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  C=0.1, decision_function_shape=ovo, gamma=1, kernel=linear, total=   0.2s
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=linear ......

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s

[CV]  C=0.1, decision_function_shape=ovo, gamma=1, kernel=linear, total=   0.3s
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=linear ......
```

Figure 8.98 : Hyperparameter Tuning for Linear Kernel

8.6.2.2. Hyperparameter Tuning for RBF Kernel

The following code snippet contains the logic of hyperparameter tuning for the RBF Kernel of SVM.

```
# Hyperparameter tuning for SVM's RBF Kernel

# defining parameter range
param_grid_rbf = {'C': [0.1, 1, 2, 3, 5, 10, 100, 1000],
                  'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'auto', 'scale'],
                  'decision_function_shape':['ovo', 'ovr'],
                  'kernel': ['rbf']}

grid_rbf = GridSearchCV(svm.SVC(), param_grid_rbf, refit = True, verbose = 2, cv=3)

# fitting the model for grid search
grid_rbf.fit(Train_X,Train_Y)

# print best parameter after tuning
print("Best parameters of RBF Kernel after tuning are: ", grid_rbf.best_params_)
print("\n")
# print how our model looks after hyper-parameter tuning
print(grid_rbf.best_estimator_)
```

```
Fitting 3 folds for each of 112 candidates, totalling 336 fits
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=rbf .........

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  C=0.1, decision_function_shape=ovo, gamma=1, kernel=rbf, total=   0.4s
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=rbf .........

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s

[CV]  C=0.1, decision_function_shape=ovo, gamma=1, kernel=rbf, total=   0.3s
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=rbf .........
```

Figure 8.99 : Hyperparameter Tuning for RBF Kernel

8.6.2.3. Hyperparameter Tuning for Polynomial Kernel

The following code snippet contains the logic of hyperparameter tuning for the Polynomial Kernel of SVM.

```
# Hyperparameter tuning for SVM's Polynomial Kernel

# defining parameter range
param_grid_poly = {'C': [0.1, 1, 2, 3, 5, 10, 100, 1000],
                   'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'auto', 'scale'],
                   'decision_function_shape':['ovo', 'ovr'],
                   'degree': [1, 1.5, 3],
                   'coef0': [0.01, 0.1, 1, 10],
                   'kernel': ['poly']}

grid_poly = GridSearchCV(svm.SVC(), param_grid_poly, refit = True, verbose = 2, cv=3)

# fitting the model for grid search
grid_poly.fit(Train_X, Train_Y)

# print best parameter after tuning
print("Best parameters of Polynomial Kernel after tuning are: ", grid_poly.best_params_)
print("\n")
# print how our model looks after hyper-parameter tuning
print(grid_poly.best_estimator_)
```

```
Fitting 3 folds for each of 1344 candidates, totalling 4032 fits
[CV] C=0.1, coef0=0.01, decision_function_shape=ovo, degree=1, gamma=1, kernel=poly
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  C=0.1, coef0=0.01, decision_function_shape=ovo, degree=1, gamma=1, kernel=poly, total=   0.2s
[CV] C=0.1, coef0=0.01, decision_function_shape=ovo, degree=1, gamma=1, kernel=poly
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
```

Figure 8.100 : Hyperparameter Tuning for Polynomial Kernel

8.6.2.4. Hyperparameter Tuning for Sigmoid Kernel

The following code snippet contains the logic of hyperparameter tuning for the Sigmoid Kernel of SVM.

```
# Hyperparameter tuning for SVM's Sigmoid Kernel

# defining parameter range
param_grid_sigmoid = {'C': [0.1, 1, 2, 3, 5, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'auto', 'scale'],
              'decision_function_shape':['ovo', 'ovr'],
              'coef0': [0.01, 0.1, 1, 10],
              'kernel': ['sigmoid']}

grid_sigmoid = GridSearchCV(svm.SVC(), param_grid_sigmoid, refit = True, verbose = 2, cv=3)

# fitting the model for grid search
grid_sigmoid.fit(Train_X, Train_Y)

# print best parameter after tuning
print("Best parameters of Sigmoid Kernel after tuning are: ", grid_sigmoid.best_params_)
print("\n")
# print how our model looks after hyper-parameter tuning
print(grid_sigmoid.best_estimator_)
```

```
Fitting 3 folds for each of 448 candidates, totalling 1344 fits
[CV] C=0.1, coef0=0.01, decision_function_shape=ovo, gamma=1, kernel=sigmoid

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  C=0.1, coef0=0.01, decision_function_shape=ovo, gamma=1, kernel=sigmoid, total=   0.3s
[CV] C=0.1, coef0=0.01, decision_function_shape=ovo, gamma=1, kernel=sigmoid

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s

[CV]  C=0.1, coef0=0.01, decision_function_shape=ovo, gamma=1, kernel=sigmoid, total=   0.3s
```

Figure 8.101 : Hyperparameter Tuning for Sigmoid Kernel

8.6.2.5. Hyperparameter Tuning for Precomputed Kernel

The following code snippet contains the logic of hyperparameter tuning for the Precomputed Kernel of SVM.

```
# Hyperparameter tuning for SVM's Precomputed Kernel

Train_X = Train_X.todense()
Test_X = Test_X.todense()
gram_train = np.dot(Train_X, Train_X.T)
gram_test=np.dot(Test_X, Train_X.T)

# defining parameter range
param_grid_precom = {'C': [0.1, 1, 2, 3, 5, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'auto', 'scale'],
              'decision_function_shape':['ovo', 'ovr'],
              'kernel': ['precomputed']}

grid_precom = GridSearchCV(svm.SVC(), param_grid_precom, refit = True, verbose = 2, cv=3)

# fitting the model for grid search
grid_precom.fit(gram_train, Train_Y)

# print best parameter after tuning
print("Best parameters of Precomputed Kernel after tuning are: ", grid_precom.best_params_)
print("\n")
# print how our model looks after hyper-parameter tuning
print(grid_precom.best_estimator_)
```

```
Fitting 3 folds for each of 112 candidates, totalling 336 fits
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=precomputed .
[CV]  C=0.1, decision_function_shape=ovo, gamma=1, kernel=precomputed, total=   0.2s
[CV] C=0.1, decision_function_shape=ovo, gamma=1, kernel=precomputed .

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

Figure 8.102 : Hyperparameter Tuning for Precomputed Kernel

8.6.3. Building the SVM Models with Best Hyperparameters for each Kernel

After completing hyperparameter tuning, SVM Models were built for each Kernel with the combination of hyperparameters obtained as the results of hyperparameter tuning.

After building models relevant for each kernel Accuracy and F1- Score values were gained.

Since the RBF Kernel promised the accuracy of 87.02%, it was used in further predictions of the SVM Model.

8.6.3.1. Building the SVM Model with Best Hyperparameters for Linear Kernel

The following code snippet contains the logic of building the SVM Model with optimal combination of hyperparameters for the Linear Kernel of SVM.

```
[Parallel(n_jobs=1)]: Done 336 out of 336 | elapsed:  2.1min finished

Best parameters of Linear Kernel after tuning are:  {'C': 1, 'decision_function_shape': 'ovo', 'gamma': 1, 'kernel':
'linear'}


SVC(C=1, decision_function_shape='ovo', gamma=1, kernel='linear')
```

```
# SVM Classifier with Linear Kernel with shuffling
linear = svm.SVC(C=1, decision_function_shape='ovo', gamma=1, kernel='linear').fit(Train_X, Train_Y)

# save the model
filename = 'Linear_SVM_model.sav'
pickle.dump(linear, open(filename, 'wb'))

# Accuracy and F1-Score predictions
linear_pred = linear.predict(Test_X)
linear_accuracy = accuracy_score(Test_Y, linear_pred)
linear_f1 = f1_score(Test_Y, linear_pred, average='weighted')
print('Accuracy (Linear Kernel): ', "%.2f" % (linear_accuracy*100))
print('F1 (Linear Kernel): ', "%.2f" % (linear_f1*100))
```

```
Accuracy (Linear Kernel):  85.58
F1 (Linear Kernel):  85.77
```

Figure 8.103 : Building the SVM Model with optimal combination of hyperparameters for the Linear Kernel

8.6.3.2. Building the SVM Model with Best Hyperparameters for RBF Kernel

The following code snippet contains the logic of building the SVM Model with optimal combination of hyperparameters for the RBF Kernel of SVM.

```
[Parallel(n_jobs=1)]: Done 336 out of 336 | elapsed:  1.8min finished

Best parameters of RBF Kernel after tuning are:  {'C': 2, 'decision_function_shape': 'ovo', 'gamma': 'scale', 'kernel
': 'rbf'}


SVC(C=2, decision_function_shape='ovo')
```

```
# SVM Classifier with RBF Kernel with shuffling
rbf = svm.SVC(kernel='rbf', C=2, decision_function_shape='ovo', gamma='scale').fit(Train_X, Train_Y)

# save the model
filename = 'RBF_SVM_model.sav'
pickle.dump(rbf, open(filename, 'wb'))

# Accuracy and F1-Score predictions
rbf_pred = rbf.predict(Test_X)
rbf_accuracy = accuracy_score(Test_Y, rbf_pred)
rbf_f1 = f1_score(Test_Y, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
```

```
Accuracy (RBF Kernel):  87.02
F1 (RBF Kernel):  87.11
```

Figure 8.104 : Building the SVM Model with optimal combination of hyperparameters for the RBF Kernel

8.6.3.3. Building the SVM Model with Best Hyperparameters for Polynomial Kernel

The following code snippet contains the logic of building the SVM Model with optimal combination of hyperparameters for the Polynomial Kernel of SVM.

```
[Parallel(n_jobs=1)]: Done 4032 out of 4032 | elapsed: 22.2min finished

Best parameters of Polynomial Kernel after tuning are:  {'C': 1, 'coef0': 0.01, 'decision_function_shape': 'ovo', 'de
gree': 1, 'gamma': 'scale', 'kernel': 'poly'}


SVC(C=1, coef0=0.01, decision_function_shape='ovo', degree=1, kernel='poly')
```

```
# SVM Classifier with Polynomial Kernel with shuffling
poly = svm.SVC(C=1, coef0=0.01, decision_function_shape='ovo', degree=1, kernel='poly', gamma='scale').fit(Train_X,
                                                                                                           Train_Y)

# save the model
filename = 'Polynomial_SVM_model.sav'
pickle.dump(poly, open(filename, 'wb'))

# Accuracy and F1-Score predictions
poly_pred = poly.predict(Test_X)
poly_accuracy = accuracy_score(Test_Y, poly_pred)
poly_f1 = f1_score(Test_Y, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
```

```
Accuracy (Polynomial Kernel):  85.74
F1 (Polynomial Kernel):  85.92
```

Figure 8.105 : Building the SVM Model with optimal combination of hyperparameters for the Polynomial Kernel

8.6.3.4. Building the SVM Model with Best Hyperparameters for Sigmoid Kernel

The following code snippet contains the logic of building the SVM Model with optimal combination of hyperparameters for the Sigmoid Kernel of SVM.

```
[Parallel(n_jobs=1)]: Done 1344 out of 1344 | elapsed:  6.6min finished

Best parameters of Sigmoid Kernel after tuning are:  {'C': 10, 'coef0': 0.01, 'decision_function_shape': 'ovo', 'gamm
a': 0.1, 'kernel': 'sigmoid'}


SVC(C=10, coef0=0.01, decision_function_shape='ovo', gamma=0.1,
    kernel='sigmoid')
```

```
# SVM Classifier with Sigmoid Kernel with shuffling
sigmoid = svm.SVC(C=10, coef0=0.01, decision_function_shape='ovo', gamma=0.1, kernel='sigmoid').fit(Train_X, Train_Y)

# save the model
filename = 'Sigmoid_SVM_model.sav'
pickle.dump(sigmoid, open(filename, 'wb'))

# Accuracy and F1-Score predictions
sigmoid_pred = sigmoid.predict(Test_X)
sigmoid_accuracy = accuracy_score(Test_Y, sigmoid_pred)
sigmoid_f1 = f1_score(Test_Y, sigmoid_pred, average='weighted')
print('Accuracy (Sigmoid Kernel): ', "%.2f" % (sigmoid_accuracy*100))
print('F1 (Sigmoid Kernel): ', "%.2f" % (sigmoid_f1*100))
```

```
Accuracy (Sigmoid Kernel):  85.42
F1 (Sigmoid Kernel):  85.61
```

Figure 8.106 : Building the SVM Model with optimal combination of hyperparameters for the Sigmoid Kernel

8.6.3.5. Building the SVM Model with Best Hyperparameters for Precomputed Kernel

The following code snippet contains the logic of building the SVM Model with optimal combination of hyperparameters for the Precomputed Kernel of SVM.

```
[Parallel(n_jobs=1)]: Done 336 out of 336 | elapsed:  2.2min finished

Best parameters of Precomputed Kernel after tuning are:  {'C': 1, 'decision_function_shape': 'ovo', 'gamma': 1, 'kern
el': 'precomputed'}


SVC(C=1, decision_function_shape='ovo', gamma=1, kernel='precomputed')
```

```
# SVM Classifier with Precomputed Kernel with shuffling
precom = svm.SVC(kernel='precomputed', C=1, decision_function_shape='ovo', gamma=1)
Train_X = Train_X.todense()
Test_X = Test_X.todense()
gram_train = np.dot(Train_X, Train_X.T)
precom.fit(gram_train, Train_Y)

# save the model
filename = 'Precomputed_SVM_model.sav'
pickle.dump(precom, open(filename, 'wb'))

# Accuracy and F1-Score predictions

gram_test=np.dot(Test_X, Train_X.T)
precom_pred = precom.predict(gram_test)
precom_accuracy = accuracy_score(Test_Y, precom_pred)
precom_f1 = f1_score(Test_Y, precom_pred, average='weighted')
print('Accuracy (Precomputed Kernel): ', "%.2f" % (precom_accuracy*100))
print('F1 (Preomputed Kernel): ', "%.2f" % (precom_f1*100))
```

```
Accuracy (Precomputed Kernel):  85.58
F1 (Preomputed Kernel):  85.77
```

Figure 8.107 : Building the SVM Model with optimal combination of hyperparameters for the Precomputed Kernel

9. Evaluation

Since the Exploration of Useful Comments in Stack Overflow was based on building a Classification Model which categorizes Useful Comments with their respective Useful Comment Category (Standard Comment Categories that are being listed in the Commenting Guidelines of Stack Overflow), the evaluation of this model was carried out using the Holdout method, Confusion Matrix and the Classification Report.

At first it was necessary to gain insight of the instances belonging to each category in Test data.

```
# Gaining insight of instances belonging to each category in Test set
Test_Y
count_request_clarificaton=0
count_relevant_information=0
count_constructive_criticism=0
for label in Test_Y:
    if label==0:
        count_constructive_criticism+=1
    if label==1:
        count_relevant_information+=1
    if label==2:
        count_request_clarificaton+=1

print("Count of Request Clarification instances in Test set is, ", count_request_clarificaton)
print("Count of Relevant Information in Test set is, ", count_relevant_information)
print("Count of Constructive Criticism instances in Test set is, ", count_constructive_criticism)
```

```
Count of Request Clarification instances in Test set is,  202
Count of Relevant Information in Test set is,  218
Count of Constructive Criticism instances in Test set is,  204
```

Figure 9.1 : Data count of each comment category in Test Set

After this calculation, it was apparent that there is no clear imbalance among the comment categories in the Test Data set. This indicated that there is no such need for stratified sampling or rebalancing.

9.1. Holdout method based Evaluation

The Holdout method is the commonly used method to evaluate a classification model. In the Holdout Method the data set is divided into 2 parts such as train set and test set. Train set contains 80% of the data while the Test set contains 20% of the data. The Train set is utilized to train the data and the Test set is utilized to test the predictive power of the implemented Classification model. Classification Accuracy and F1-Score is gained as the metric of evaluation in the Holdout method.

For the initial SVM Model the Evaluation was performed with the holdout method.

```
SVM_Model_pred = SVM_Model.predict(Test_X)

SVM_Model_accuracy = accuracy_score(Test_Y, SVM_Model_pred)
SVM_Model_f1 = f1_score(Test_Y, SVM_Model_pred, average='weighted')
```

Figure 9.2 : Logic of predicting accuracy and f1-score based on holdout method for the initial SVM Model

```
Accuracy :  87.02
F1 :  87.21
```

Figure 9.3 : Accuracy and F1-Score values for the initial SVM Model based on Holdout Method

Moreover, the evaluations of each SVM Model implemented with Linear, RBF, Polynomial, Sigmoid and Precomputed Kernels after including optimal parameters gained through hyperparameter tuning was also performed using the holdout method.

```
# Accuracy and F1-Score predictions
linear_pred = linear.predict(Test_X)
linear_accuracy = accuracy_score(Test_Y, linear_pred)
linear_f1 = f1_score(Test_Y, linear_pred, average='weighted')
print('Accuracy (Linear Kernel): ', "%.2f" % (linear_accuracy*100))
print('F1 (Linear Kernel): ', "%.2f" % (linear_f1*100))
```

```
Accuracy (Linear Kernel):  85.58
F1 (Linear Kernel):  85.77
```

Figure 9.4 : Accuracy and F1-Score values for the Linear SVM Model based on Holdout Method

```
# Accuracy and F1-Score predictions
rbf_pred = rbf.predict(Test_X)
rbf_accuracy = accuracy_score(Test_Y, rbf_pred)
rbf_f1 = f1_score(Test_Y, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
```

```
Accuracy (RBF Kernel):  87.02
F1 (RBF Kernel):  87.11
```

Figure 9.5 : Accuracy and F1-Score values for the RBF SVM Model based on Holdout Method

```
# Accuracy and F1-Score predictions
poly_pred = poly.predict(Test_X)
poly_accuracy = accuracy_score(Test_Y, poly_pred)
poly_f1 = f1_score(Test_Y, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
```

```
Accuracy (Polynomial Kernel):  85.74
F1 (Polynomial Kernel):  85.92
```

Figure 9.6 : Accuracy and F1-Score values for the Polynomial SVM Model based on Holdout Method

```
# Accuracy and F1-Score predictions
sigmoid_pred = sigmoid.predict(Test_X)
sigmoid_accuracy = accuracy_score(Test_Y, sigmoid_pred)
sigmoid_f1 = f1_score(Test_Y, sigmoid_pred, average='weighted')
print('Accuracy (Sigmoid Kernel): ', "%.2f" % (sigmoid_accuracy*100))
print('F1 (Sigmoid Kernel): ', "%.2f" % (sigmoid_f1*100))
```

```
Accuracy (Sigmoid Kernel):  85.42
F1 (Sigmoid Kernel):  85.61
```

Figure 9.7 : Accuracy and F1-Score values for the Sigmoid SVM Model based on Holdout Method

```
# Accuracy and F1-Score predictions

gram_test=np.dot(Test_X, Train_X.T)
precom_pred = precom.predict(gram_test)
precom_accuracy = accuracy_score(Test_Y, precom_pred)
precom_f1 = f1_score(Test_Y, precom_pred, average='weighted')
print('Accuracy (Precomputed Kernel): ', "%.2f" % (precom_accuracy*100))
print('F1 (Preomputed Kernel): ', "%.2f" % (precom_f1*100))
```

```
Accuracy (Precomputed Kernel):  85.58
F1 (Preomputed Kernel):  85.77
```

Figure 9.8 : Accuracy and F1-Score values for the Precomputed SVM Model based on Holdout Method

Results of the Holdout Method can be summarized as follows.

Kernel of SVM	**Accuracy**	**F1-Score**
Linear	85.58	85.77
RBF	87.02	87.11
Polynomial	85.74	85.92
Sigmoid	85.42	85.61
Precomputed	85.58	85.77

Table 9.1: Summarized results of the Holdout Method

According to the above results, SVM Model with the RBF Kernel can be identified as the best Classification Model as it promised the highest Accuracy of 87.02 and highest F1-Score of 87.11.

9.2. Confusion Matrix based Evaluation

When regarding the Confusion Matrix, for the purpose of gaining insights of the performance of the classification model the confusion matrix can be used. The information gained from the confusion matrix can be used to determine the usefulness of the classification model. As a result important metrics such as accuracy, precision and recall can be determined.

TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative

- Accuracy = (TP + TN)/(TP + TN + FP + FN)
- Precision = TP/(TP+FP)
- Recall = TP/(TP+FN)

		True Classes	
		Positive	Negative
Predicted Classes	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 9.2: Logic of a typical 2x2 Confusion Matrix

Since Useful Comments fall into 3 categories 3x3 Confusion Matrix is used for evaluation. In the 3x3 Confusion Matrix non-diagonal values are identified as misclassified.

		Predicted Classes		
		Constructive Criticism (CC)	Relevant Information (RI)	Request Clarification (RQ)
True Classes	Constructive Criticism (CC)	True-CC (Actual and Predicted are the same)	False-RI	False-RQ
	Relevant Information (RI)	False- CC	True-RI (Actual and Predicted	False-RQ

			are the same)	
	Request Clarification (RQ)	False-CC	False-RI	True-RQ (Actual and Predicted are the same)

Table 9.3: The Logic of the utilized 3x3 Confusion Matrix

The confusion matrix was drawn for the initial SVM Model which was built before hyperparameter tuning. The logic utilized and the results obtained are as follows.

```
from sklearn.metrics import confusion_matrix

confusion_SVM = confusion_matrix(Test_Y, SVM_Model_pred)
print('\nConfusion Matrix of the Initial SVM Model\n')
print(confusion_SVM)
```

Figure 9.9 : The Logic of Confusion Matrix for the initial SVM Model

```
Confusion Matrix of the Initial SVM Model

[[173   5  26]
 [  1 186  31]
 [  8  10 184]]
```

Figure 9.10 : Confusion Matrix Obtained for the Initial SVM Model

As per the above result the initial SVM Model had a total number of correctly classified comments of 543.

Afterwards, the confusion matrices were plotted for each SVM Model with distinct kernels and optimal combination of hyperparameters obtained after hyperparameter tuning.

```
# Confusion Matrix Logic for SVM's Kernels

from sklearn.metrics import confusion_matrix

confusion_linear = confusion_matrix(Test_Y, linear_pred)
print('\nConfusion Matrix of Linear Kernel of SVM\n')
print(confusion_linear)

confusion_rbf = confusion_matrix(Test_Y, rbf_pred)
print('\nConfusion Matrix of RBF Kernel of SVM\n')
print(confusion_rbf)

confusion_poly = confusion_matrix(Test_Y, poly_pred)
print('\nConfusion Matrix of Polynomial Kernel of SVM\n')
print(confusion_poly)

confusion_sigmoid = confusion_matrix(Test_Y, sigmoid_pred)
print('\nConfusion Matrix of Sigmoid Kernel of SVM\n')
print(confusion_sigmoid)

confusion_precom = confusion_matrix(Test_Y, precom_pred)
print('\nConfusion Matrix of Precomputed Kernel of SVM\n')
print(confusion_precom)
```

Figure 9.11 : The Logic of Confusion Matrix for each SVM Model built with respective distinct kernels

```
Confusion Matrix of Linear Kernel of SVM

[[175   5  24]
 [  4 174  40]
 [  8   9 185]]

Confusion Matrix of RBF Kernel of SVM

[[179   6  19]
 [  5 185  28]
 [ 13  10 179]]

Confusion Matrix of Polynomial Kernel of SVM

[[176   5  23]
 [  4 174  40]
 [  8   9 185]]

Confusion Matrix of Sigmoid Kernel of SVM

[[175   5  24]
 [  4 174  40]
 [  9   9 184]]

Confusion Matrix of Precomputed Kernel of SVM

[[175   5  24]
 [  4 174  40]
 [  8   9 185]]
```

Figure 9.12 : Output Confusion Matrices for the basic logic of gaining Confusion Matrices

```
# Plotting of Confusion Matrix for SVM's Kernels

import itertools

def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

Figure 9.13 : Function for plotting confusion matrix for SVM’s Kernels

```
    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()

plot_confusion_matrix(cm           = np.array(confusion_linear),
                      normalize    = False,
                      target_names = ['Constructive Criticism', 'Relevant Information', 'Request Clarification'],
                      title        = "Confusion Matrix for Linear Kernel of SVM")

plot_confusion_matrix(cm           = np.array(confusion_rbf),
                      normalize    = False,
                      target_names = ['Constructive Criticism', 'Relevant Information', 'Request Clarification'],
                      title        = "Confusion Matrix for RBF Kernel of SVM")
```

Figure 9.14 : Plotting confusion matrices for SVM's Linear and RBF Kernels

```
plot_confusion_matrix(cm           = np.array(confusion_poly),
                      normalize    = False,
                      target_names = ['Constructive Criticism', 'Relevant Information', 'Request Clarification'],
                      title        = "Confusion Matrix for Polynomial Kernel of SVM")

plot_confusion_matrix(cm           = np.array(confusion_sigmoid),
                      normalize    = False,
                      target_names = ['Constructive Criticism', 'Relevant Information', 'Request Clarification'],
                      title        = "Confusion Matrix for Sigmoid Kernel of SVM")

plot_confusion_matrix(cm           = np.array(confusion_precom),
                      normalize    = False,
                      target_names = ['Constructive Criticism', 'Relevant Information', 'Request Clarification'],
                      title        = "Confusion Matrix for Precomputed Kernel of SVM")
```

Figure 9.15 : Plotting confusion matrices for SVM's Polynomial, Sigmoid and Precomputed Kernels

The results obtained after plotting the confusion matrices for each kernel in SVM is shown in plots below.

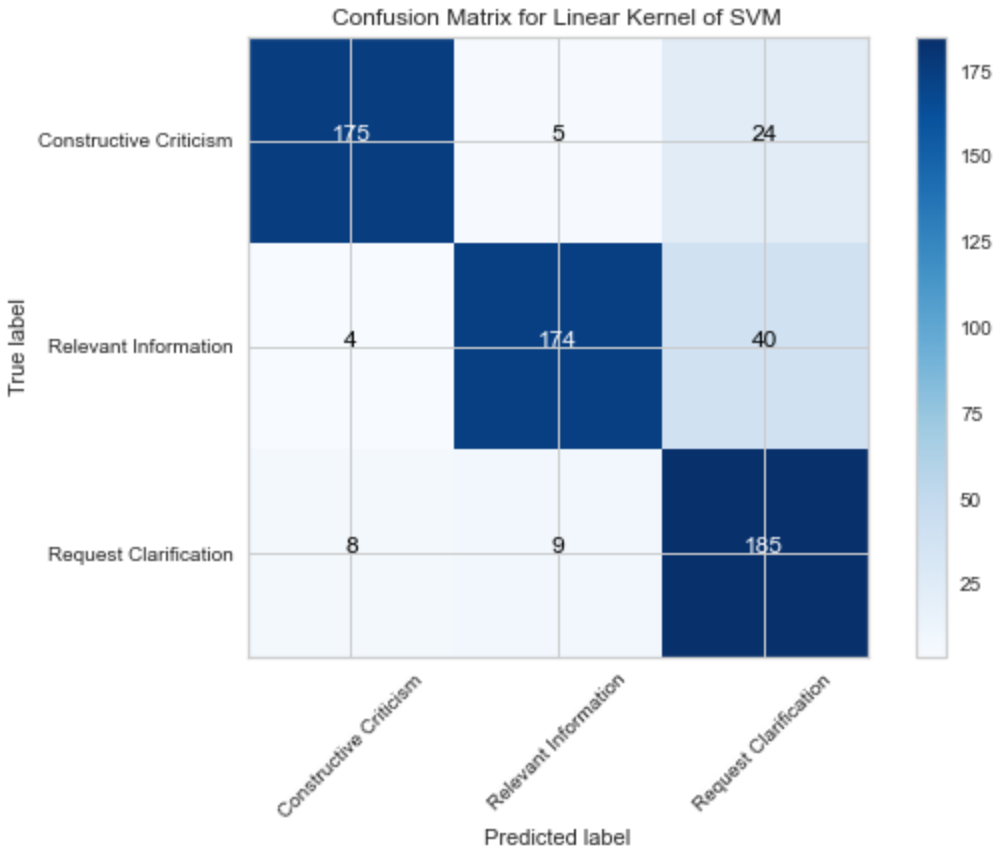


Figure 9.16 : Confusion Matrix for Linear Kernel of SVM

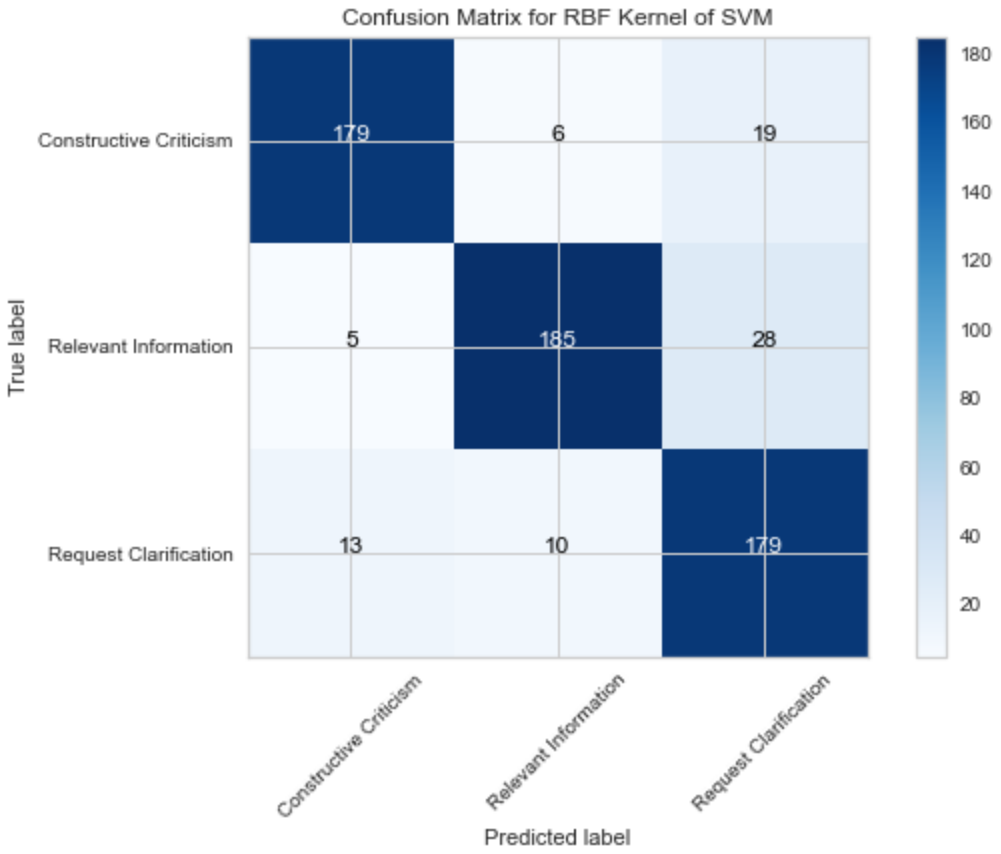


Figure 9.17 : Confusion Matrix for RBF Kernel of SVM

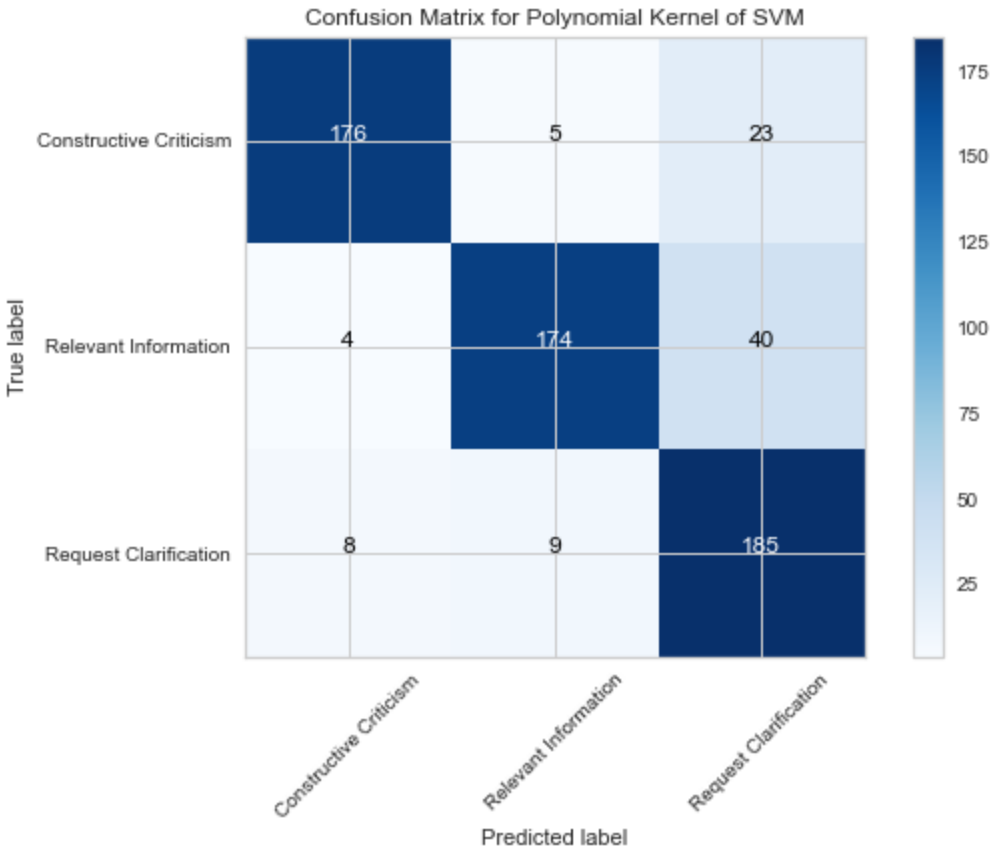


Figure 9.18 : Confusion Matrix for Polynomial Kernel of SVM

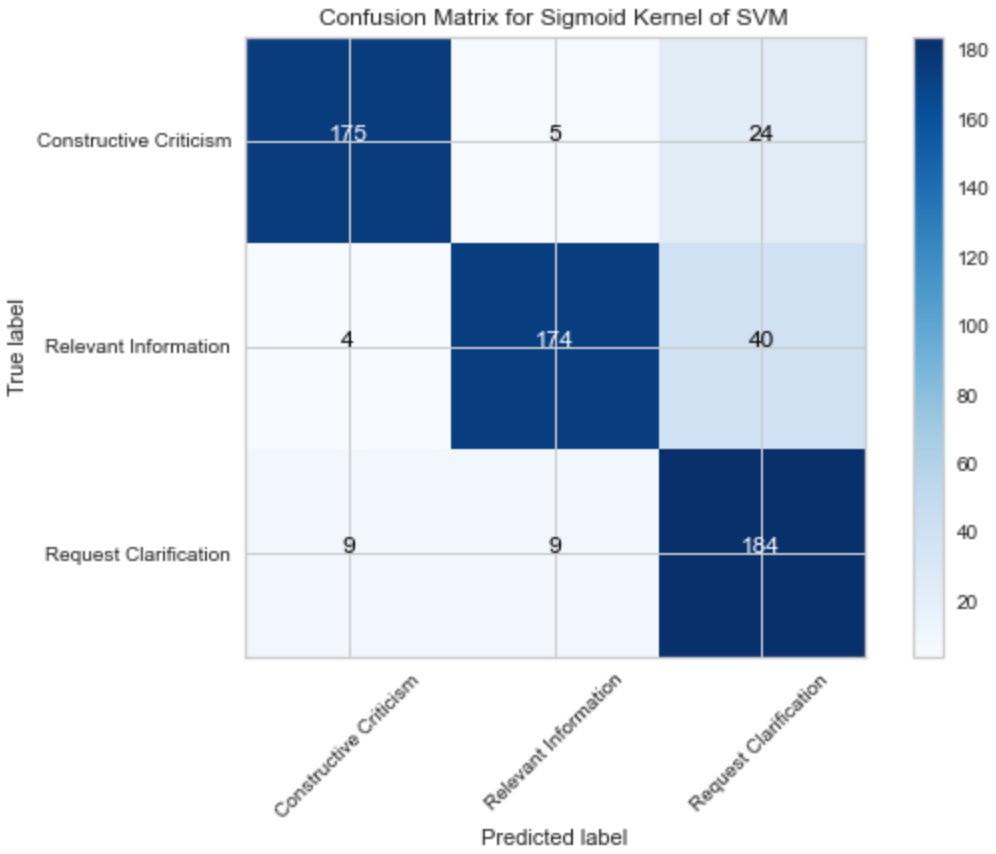


Figure 9.19 : Confusion Matrix for Sigmoid Kernel of SVM

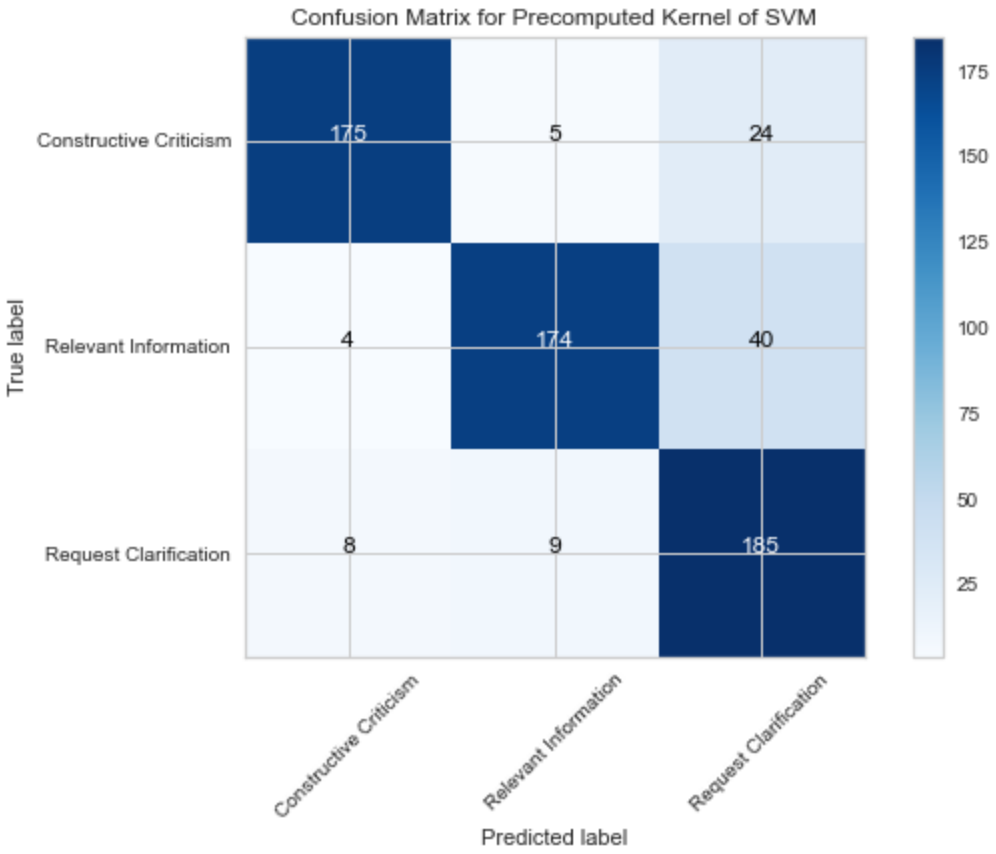


Figure 9.20 : Confusion Matrix for Precomputed Kernel of SVM

The results obtained through the Confusion Matrices relevant to each SVM Model can be summarized as follows.

SVM Models with distinct kernels	**Number of comments in each Useful Comment Category where True Label = Predicted Label**			**Total number of comments where True Label = Predicted Label**
	Constructive Criticism	**Relevant Information**	**Request Clarification**	
Linear Kernel Model	175	174	185	534
RBF Kernel	179	185	179	543

Model				
Polynomial Kernel Model	176	174	185	535
Sigmoid Kernel Model	175	174	184	533
Precomputed Kernel Model	175	174	185	534

Table 9.4 : Summarized Results obtained through the 3x3 Confusion Matrix

As per the above results, the SVM Model with the RBF Kernel can be identified as the best model with highest performance as it promised the highest number of correctly predicted instances where true label equals to predicted label i.e. 543.

9.3. Classification Report based Evaluation

The quality of the predictions relevant to a Classification algorithm is often measured by a Classification Report. It provides the main classification metrics based on each class. True Positives, True Negatives, False Positives and False Negatives are utilized for the purpose of predicting metrics in a Classification Report.

- Precision is the ability of a specific classifier to recognize only the correct instances of each class.
- Recall is the ability of a specific classifier to recognize all correct instances per class.
- F1-Score is the weighted harmonic mean of recall and precision, which is then normalized between 0 and 1. Since high precision and recall scores are important, a high F1-Score is often recognized as useful.
- Support is the actual occurrence of a specific class in the test data set.

Along with the Classification report metrics relevant to macro average, micro average and weighted average are also being calculated for precision, recall and f1-score.

- The macro average measure calculates a specific metric for each class to calculate the respective unweighted average.
- The micro average measure calculates the average of a specific metric from the aggregate contributions of all classes.
- The weighted average calculates the metrics for each label, and determines their average weighted by support (the number of true occurrences for each label).

Initially, the Classification report was obtained for the initial SVM Model which was built without including any parameters. The Logic utilized and the results obtained are as follows.

```
print('Micro Precision: {:.2f}'.format(precision_score(Test_Y, SVM_Model_pred, average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(Test_Y, SVM_Model_pred, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(Test_Y, SVM_Model_pred, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(Test_Y, SVM_Model_pred, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(Test_Y, SVM_Model_pred, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(Test_Y, SVM_Model_pred, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(Test_Y, SVM_Model_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(Test_Y, SVM_Model_pred, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(Test_Y, SVM_Model_pred, average='weighted')))
```

Figure 9.21 : Calculating Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score for the Initial SVM Model

```
print('\n-----Classification Report related to Initial SVM Model----\n')
classificationReport = classification_report(Test_Y, SVM_Model_pred, target_names=['Class 1', 'Class 2', 'Class 3'])
print(classificationReport)
```

Figure 9.22 : Logic of computing the Classification Report for the Initial SVM Model

```
Micro Precision: 0.87
Micro Recall: 0.87
Micro F1-score: 0.87

Macro Precision: 0.88
Macro Recall: 0.87
Macro F1-score: 0.87

Weighted Precision: 0.88
Weighted Recall: 0.87
Weighted F1-score: 0.87
```

Figure 9.23 : Results obtained for Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score for the Initial SVM Model

```
-----Classification Report related to Initial SVM Model----

              precision    recall  f1-score   support

     Class 1       0.95      0.85      0.90       204
     Class 2       0.93      0.85      0.89       218
     Class 3       0.76      0.91      0.83       202

    accuracy                           0.87       624
   macro avg       0.88      0.87      0.87       624
weighted avg       0.88      0.87      0.87       624
```

Figure 9.24 : The Classification Report for the Initial SVM Model

Afterwards, the Classification Report was gained for each SVM Model with distinct kernels and optimal combination of hyperparameters.

```
# Classification Report related to Linear Kernel

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import classification_report

print('\nAccuracy: {:.2f}\n'.format(accuracy_score(Test_Y, linear_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(Test_Y, linear_pred, average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(Test_Y, linear_pred, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(Test_Y, linear_pred, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(Test_Y, linear_pred, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(Test_Y, linear_pred, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(Test_Y, linear_pred, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(Test_Y, linear_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(Test_Y, linear_pred, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(Test_Y, linear_pred, average='weighted')))

print('\n-----Classification Report related to Linear Kernel of SVM-----\n')
classificationReport_linear = classification_report(Test_Y, linear_pred, target_names=['Class 1', 'Class 2', 'Class 3'])
print(classificationReport_linear)
```

Figure 9.25 : Logic of Calculating Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and computing the Classification Report for the SVM Model with Linear Kernel

```
Accuracy: 0.86

Micro Precision: 0.86
Micro Recall: 0.86
Micro F1-score: 0.86

Macro Precision: 0.87
Macro Recall: 0.86
Macro F1-score: 0.86

Weighted Precision: 0.87
Weighted Recall: 0.86
Weighted F1-score: 0.86

-----Classification Report related to Linear Kernel of SVM-----

              precision    recall  f1-score   support

     Class 1       0.94      0.86      0.90       204
     Class 2       0.93      0.80      0.86       218
     Class 3       0.74      0.92      0.82       202

    accuracy                           0.86       624
   macro avg       0.87      0.86      0.86       624
weighted avg       0.87      0.86      0.86       624
```

Figure 9.26 : Results obtained for Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and the Classification Report for the SVM Model with Linear Kernel

```python
# Classification Report related to RBF Kernel
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(Test_Y, rbf_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(Test_Y, rbf_pred, average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(Test_Y, rbf_pred, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(Test_Y, rbf_pred, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(Test_Y, rbf_pred, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(Test_Y, rbf_pred, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(Test_Y, rbf_pred, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(Test_Y, rbf_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(Test_Y, rbf_pred, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(Test_Y, rbf_pred, average='weighted')))

print('\n-----Classification Report related to RBF Kernel of SVM-----\n')
classificationReport_rbf = classification_report(Test_Y, rbf_pred, target_names=['Class 1', 'Class 2', 'Class 3'])
print(classificationReport_rbf)
```

Figure 9.27 : Logic of Calculating Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and computing the Classification Report for the SVM Model with RBF Kernel

```
Accuracy: 0.87

Micro Precision: 0.87
Micro Recall: 0.87
Micro F1-score: 0.87

Macro Precision: 0.87
Macro Recall: 0.87
Macro F1-score: 0.87

Weighted Precision: 0.87
Weighted Recall: 0.87
Weighted F1-score: 0.87

-----Classification Report related to RBF Kernel of SVM-----

              precision    recall  f1-score   support

     Class 1       0.91      0.88      0.89       204
     Class 2       0.92      0.85      0.88       218
     Class 3       0.79      0.89      0.84       202

    accuracy                           0.87       624
   macro avg       0.87      0.87      0.87       624
weighted avg       0.87      0.87      0.87       624
```

Figure 9.28 : Results obtained for Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and the Classification Report for the SVM Model with RBF Kernel

```
# Classification Report related to Polynomial Kernel

print('\nAccuracy: {:.2f}\n'.format(accuracy_score(Test_Y, poly_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(Test_Y, poly_pred, average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(Test_Y, poly_pred, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(Test_Y, poly_pred, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(Test_Y, poly_pred, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(Test_Y, poly_pred, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(Test_Y, poly_pred, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(Test_Y, poly_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(Test_Y, poly_pred, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(Test_Y, poly_pred, average='weighted')))

from sklearn.metrics import classification_report
print('\n-----Classification Report related to Polynomial Kernel of SVM-----\n')
classificationReport_poly = classification_report(Test_Y, poly_pred, target_names=['Class 1', 'Class 2', 'Class 3'])
print(classificationReport_poly)
```

Figure 9.29 : Logic of Calculating Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and computing the Classification Report for the SVM Model with Polynomial Kernel

```
Accuracy: 0.86

Micro Precision: 0.86
Micro Recall: 0.86
Micro F1-score: 0.86

Macro Precision: 0.87
Macro Recall: 0.86
Macro F1-score: 0.86

Weighted Precision: 0.87
Weighted Recall: 0.86
Weighted F1-score: 0.86

-----Classification Report related to Polynomial Kernel of SVM-----

              precision    recall  f1-score   support

     Class 1       0.94      0.86      0.90       204
     Class 2       0.93      0.80      0.86       218
     Class 3       0.75      0.92      0.82       202

    accuracy                           0.86       624
   macro avg       0.87      0.86      0.86       624
weighted avg       0.87      0.86      0.86       624
```

Figure 9.30 : Results obtained for Micro Average, Macro Average and Weighted Average for

Precision, Recall and F1-Score and the Classification Report for the SVM Model with Polynomial Kernel

```
# Classification Report related to Sigmoid Kernel
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(Test_Y, sigmoid_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(Test_Y, sigmoid_pred, average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(Test_Y, sigmoid_pred, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(Test_Y, sigmoid_pred, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(Test_Y, sigmoid_pred, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(Test_Y, sigmoid_pred, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(Test_Y, sigmoid_pred, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(Test_Y, sigmoid_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(Test_Y, sigmoid_pred, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(Test_Y, sigmoid_pred, average='weighted')))

print('\n-----Classification Report related to Sigmoid Kernel of SVM-----\n')
classificationReport_sigmoid = classification_report(Test_Y, sigmoid_pred, target_names=['Class 1', 'Class 2', 'Class 3
print(classificationReport_sigmoid)
```

Figure 9.31 : Logic of Calculating Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and computing the Classification Report for the SVM Model with Sigmoid Kernel

```
Accuracy: 0.85

Micro Precision: 0.85
Micro Recall: 0.85
Micro F1-score: 0.85

Macro Precision: 0.87
Macro Recall: 0.86
Macro F1-score: 0.86

Weighted Precision: 0.87
Weighted Recall: 0.85
Weighted F1-score: 0.86

-----Classification Report related to Sigmoid Kernel of SVM-----

              precision    recall  f1-score   support

     Class 1       0.93      0.86      0.89       204
     Class 2       0.93      0.80      0.86       218
     Class 3       0.74      0.91      0.82       202

    accuracy                           0.85       624
   macro avg       0.87      0.86      0.86       624
weighted avg       0.87      0.85      0.86       624
```

Figure 9.32 : Results obtained for Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and the Classification Report for the SVM Model with Sigmoid Kernel

```
# Classification Report related to Precomputed Kernel
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(Test_Y, precom_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(Test_Y, precom_pred, average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(Test_Y, precom_pred, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(Test_Y, precom_pred, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(Test_Y, precom_pred, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(Test_Y, precom_pred, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(Test_Y, precom_pred, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(Test_Y, precom_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(Test_Y, precom_pred, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(Test_Y, precom_pred, average='weighted')))

print('\n-----Classification Report related to Precomputed Kernel of SVM-----\n')
classificationReport_precom = classification_report(Test_Y, precom_pred, target_names=['Class 1', 'Class 2', 'Class 3'])
print(classificationReport_precom)
```

Figure 9.33 : Logic of Calculating Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and computing the Classification Report for the SVM Model with Precomputed Kernel

```
Accuracy: 0.86

Micro Precision: 0.86
Micro Recall: 0.86
Micro F1-score: 0.86

Macro Precision: 0.87
Macro Recall: 0.86
Macro F1-score: 0.86

Weighted Precision: 0.87
Weighted Recall: 0.86
Weighted F1-score: 0.86

-----Classification Report related to Precomputed Kernel of SVM-----

              precision    recall  f1-score   support

     Class 1       0.94      0.86      0.90       204
     Class 2       0.93      0.80      0.86       218
     Class 3       0.74      0.92      0.82       202

    accuracy                           0.86       624
   macro avg       0.87      0.86      0.86       624
weighted avg       0.87      0.86      0.86       624
```

Figure 9.34 : Results obtained for Micro Average, Macro Average and Weighted Average for Precision, Recall and F1-Score and the Classification Report for the SVM Model with Precomputed Kernel

The summarized Results obtained from the Classification Report can be indicated as follows.

SVM Kernels used in each	**Accuracy**	**Weighted Average for**	**Weighted Average for**	**Weighted Average for**

Model		Precision	Recall	F1-Score
Linear	0.86	0.87	0.86	0.86
RBF	0.87	0.87	0.87	0.87
Polynomial	0.86	0.87	0.86	0.86
Sigmoid	0.85	0.87	0.85	0.86
Precomputed	0.86	0.87	0.86	0.86

Table 9.5 : Summarized Results obtained through the Classification Report

As per the above summarized results, the weighted average for precision among all the kernels of SVM seems to be similar. However, the SVM Model with the RBF Kernel promised the highest Accuracy of 0.87, highest weighted average for Recall which of 0.87 and highest F1-Score which is 0.87 thus resulting the SVM Model with RBF Kernel to be the best Classification model for Classifying Useful Comments in Stack Overflow.

Finally a plot was depicted for the ease of visualization of Accuracies of all models which utilized distinct SVM Kernels. The Logic and the aforementioned plot is displayed below.

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.cm as cm
import itertools
```

```
# Accuracy Comparison of SVM Kernels by Visualization

x = ['Linear','RBF','Polynomial','Sigmoid','Precomputed']
y = [85.58,87.02,85.74,85.42,85.58]
colors = itertools.cycle(['b', 'g', 'r', 'c', 'm'])
plt.title("Accuracy Comparison of SVM Kernels")
plt.ylabel('Accuracy')
plt.xlabel('SVM-Kernel-details')
plt.xticks(rotation=90)
for i in range(len(y)):
    plt.bar(x[i], y[i], color=next(colors))
plt.savefig('/Users/prasadhiranasinghe/Desktop/Project_Implementation/Plots/Accuracy_Comparison.png')
plt.show()
```

Figure 9.35 : Logic of Accuracy Comparison of SVM Kernels by Visualization

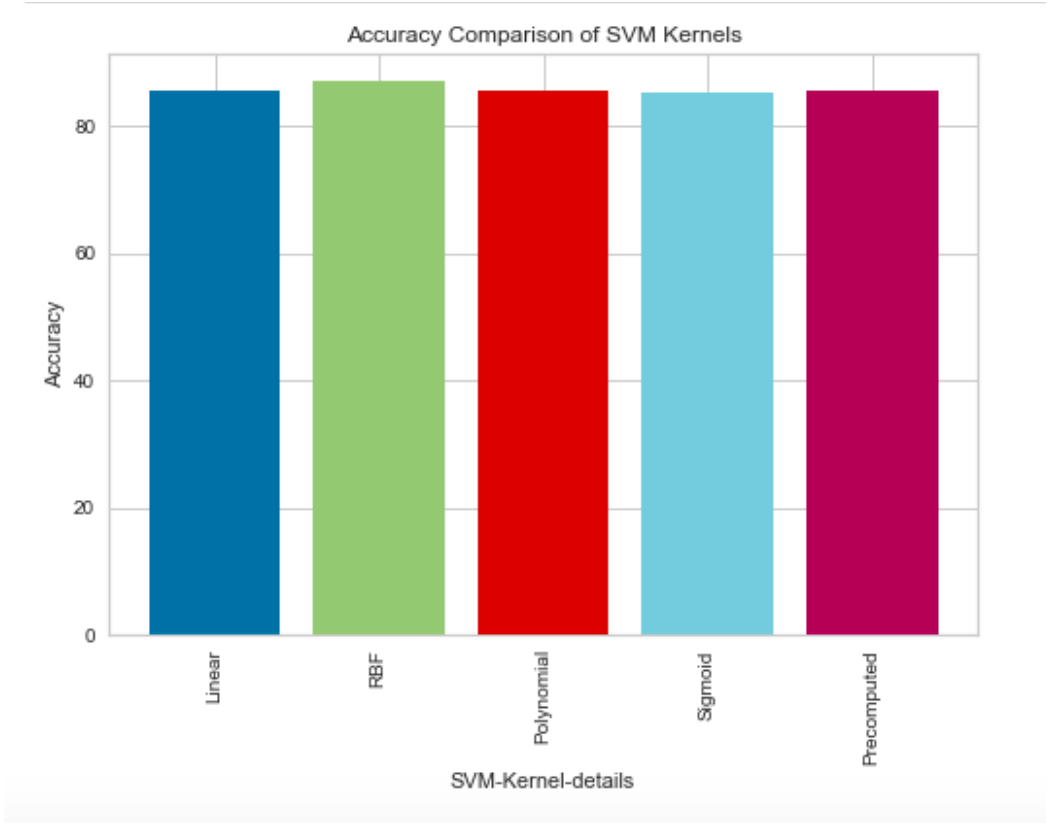


Figure 9.36 : Accuracy Comparison of SVM Kernels

As the final conclusion, the SVM Model with RBF kernel is identified as the best model with performance and accuracy for the Classification of Useful Comments in Stack Overflow.

9.4. Predictions based on the SVM Model with the Optimal Kernel

Predicting the Useful Comment Category was done using the SVM Model with RBF Kernel as it promised the highest accuracy and best performance. For these predictions random data entries of the data set were taken into consideration as depicted below. Moreover, random entries from test data and new data were also utilized in these predictions.

```
def preprocess_test_comments(text):
    Text_Without_stopWords = []
    text = text.lower()
    text = replace_urls(text)
    text = re.sub(r'\d+', '', text)
    text = text.translate(translator)
    text = remove_emoji(text)
    text = remove_emoticons(text)
    text = chatword_handler(text)
    text_tokens = nltk.word_tokenize(text)

    for word in text_tokens:
        if word not in stopwords:
            Text_Without_stopWords.append(word)

    l1 = []
    for word in Text_Without_stopWords:
        l1.append(stemmer.stem(lemmatizer.lemmatize(word,pos="v")))
    text = " ".join(l1)

    return text

def predict_Comment_Category(text):
    p_text = preprocess_test_comments(text)
    sparse_text = fitted_vectorizer.transform([text])
    Encoded_Label = rbf.predict(sparse_text)
    Categorical_Label = Encoder.inverse_transform(Encoded_Label)
    print(Categorical_Label)
```

Figure 9.37 : Function for Predicting the Useful Comment Category

The above logic contains the mechanism of decoding the encoded categorical comment labels as well as the application of cleaning the input useful comment text and vectorizing using the tf-idf vectorizer.

```
predict_Comment_Category("Can you provide your method of loading scene?")
```

```
['Request Clarification']
```

```
df[df['Text'] == "Can you provide your method of loading scene?"]
```

	Id	PostId	Score	Text	CreationDate	UserId	ContentLicense	Tags	CommentCategory	ReviewedCommentCategory	...	punctuation_%
50	93654512	53389533	3	Can you provide your method of loading scene?	11/20/2018 9:09	6396354.0	CC BY-SA 4.0	<unity3d>	Request Clarification	Request Clarification	...	2.222222

1 rows × 24 columns

Figure 9.38 : Predicting a Random Entry of Request Clarification in the Dataset

```
predict_Comment_Category("Does this answer your question? [Apache Spark: What is the equivalent implementation of RDD.g
```

```
['Relevant Information']
```

```
df[df['Text'] == "Does this answer your question? [Apache Spark: What is the equivalent implementation of RDD.groupByKe
```

	Id	PostId	Score	Text	CreationDate	UserId	ContentLicense	Tags	CommentCategory	ReviewedCommentCategory	...	punctuatio
91	107698040	60869102	3	Does this answer your question? [Apache Spark:...	3/26/2020 16:35	10938362.0	CC BY-SA 4.0	<apache-spark>	Relevant Information	Relevant Information	...	12.062

1 rows × 24 columns

Figure 9.39: Predicting a Random Entry of Relevant Information in the Dataset

```
predict_Comment_Category("I think the documentation part is the right place where you can post this. Actually you don't
```

```
['Constructive Criticism']
```

```
df[df['Text'] == "I think the documentation part is the right place where you can post this. Actually you don't have a
```

	Id	PostId	Score	Text	CreationDate	UserId	ContentLicense	Tags	CommentCategory	ReviewedCommentCategory	...	punctu
85	78095393	45566463	3	I think the documentation part is the right pl...	8/8/2017 12:19	6452829.0	CC BY-SA 3.0	<javascript> <node.js> <event-loop>	Constructive Criticism	Constructive Criticism	...	

1 rows × 24 columns

Figure 9.40: Predicting a Random Entry of Constructive Criticism in the Dataset

```
predict_Comment_Category("pleas includ full code do")
```

```
['Request Clarification']
```

```
Test_data[Test_data['text_final'] == "pleas includ full code do"]
```

	text_final	label
729	pleas includ full code do	Request Clarification

Figure 9.41: Predicting a Random Entry of Request Clarification in the Test Dataset

```
predict_Comment_Category("pleas improv question pertin inform discuss uncov chat stack overflow question help futur vis
```

```
['Constructive Criticism']
```

```
Test_data[Test_data['text_final'] == "pleas improv question pertin inform discuss uncov chat stack overflow question he
```

	text_final	label
2622	pleas improv question pertin inform discuss un...	Constructive Criticism

Figure 9.42: Predicting a Random Entry of Constructive Criticism in the Test Dataset

```
predict_Comment_Category("possibl duplic aspnet mvc viewmodel dropdownlist datalink")
```

```
['Relevant Information']
```

```
Test_data[Test_data['text_final'] == "possibl duplic aspnet mvc viewmodel dropdownlist datalink"]
```

	text_final	label
499	possibl duplic aspnet mvc viewmodel dropdownli...	Relevant Information

Figure 9.43: Predicting a Random Entry of Relevant Information in the Test Dataset

```
# Random Prediction for Request Clarification - encoding 2
predict_Comment_Category("what do you mean not working smoothly?")
```

```
['Request Clarification']
```

```
# Random Prediction for Relevant Information - encoding 1
predict_Comment_Category("This is a duplicate of https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
```

```
['Relevant Information']
```

```
# Random Prediction for Request Clarification - encoding 2
predict_Comment_Category("Please format your question")
```

```
['Constructive Criticism']
```

Figure 9.44: Random predictions relevant to all three comment categories done for external new data

10. Conclusion and Further Work

The module which Explores Useful Comments in Stack Overflow classifies the Comments in Stack Overflow which adheres to Stack Overflow's commenting guidelines into the respective standard comment categories. Initially 6164 comments were sampled using the simple random sampling technique and a qualitative analysis was performed to label and separate Useful Comments from the Comments which do not follow the Commenting guidelines of Stack Overflow. The review was performed to the resulting 3587 comments which were initially identified as Useful Comments and 3120 Comments were finally labeled and identified as Useful Comments after the review. These 3120 Useful Comments were used to build the Multi Class Classification Model in which Preprocessing, Feature Extraction, Training of the Model and Evaluation was done. Feature Engineering resulted that there exists no any features within the specified 9 features that can be combined with the textual features for the purpose of implementing the Classification Model. Preprocessing contained Lowercasing, Replacement of URLs with a keyword, Removal of Punctuation, numbers, emojis, emoticons and Stop words, Tokenization, Replacement of Chat Words, Stemming and Lemmatization. TF-IDF features and N-grams features were used as features after performing feature extraction. Support Vector Machine(SVM) was used as the Classification Algorithm and 5 different kernels were used for training purposes. Evaluation was conducted using the Holdout Method, Confusion Matrix and Classification Report. Radial Basis Function(RBF) was identified as the best Kernel when regarding the Exploration of Useful Comments in Stack Overflow as it promised the highest accuracy. Since this module performs Multi Class Classification, further improvements for this module can be to utilize Multi Label Classification for the useful comments which belong to multiple useful comment categories.

Reference

[1] Mohammad Masudar Rahman, Chanchal K. Roy and Iman Keivanloo, *Recommending Insightful Comments For Source Code Using Crowdsourced Knowledge*. Bremen, Germany: IEEE, 2015. pp.81-90.

[2] Abhishek Soni and Sarah Nadi, *Analyzing Comment-Induced Updates On Stack Overflow*.

[3] Nicole Novielli, Fabrio Calefato and Filippo Lanubile, *A Gold Standard for Emotion Annotation in Stack Overflow*. Gothenburg, Sweden: MSR, 2018, pp. 1-4.

[4] E. Wong, J. Yang and L. Tan, *AutoComment: Mining Question and Answer Sites for Automatic Comment Generation*. Palo Alto, USA: IEEE, 2013, pp. 562-567.

[5] C. Vassallo, S. Panichella, M. Di Penta and G. Canfora, *CODES: mining sourCe cOde Descriptions from developErs diScussions*. Hyderabad, 2014, pp. 106-109.

[6] H. Zhang, S. Wang, T. Chen and A. Hassan, *Reading Answers on Stack Overflow: Not Enough!*. IEEE Transactions on Software Engineering, 2019, pp. 1-15.

[7] Sengupta and C. Haythornthwaite, Learning with comments: *An analysis of comments and community on Stack Overflow*. Hawaii: 53rd Hawaii International Conference on System Sciences, 2020, pp. 2898 - 2907.

[8] Choetkiertikul, M., Avery, D., Khanh Dam, H., Tran, T. and Ghose, A., 2015. *Who Will Answer My Question On Stack Overflow?*. ASWEC.

[9] A. L Sulke and A. S Varude, *Classification of Online Pernicious Comments using Machine Learning*. Pune, India: International Journal for Scientific Research & Development, 2019, pp. 363-366.

[10] A. Sri Manasa Venigalla, C. S. Lakkundi and S. Chimalakonda, *SOTagger - Towards Classifying Stack Overflow Posts through Contextual Tagging*.

[11] S. Beyer, C. Macho, M. Di Penta and M. Pinzger, *What kind of questions do developers ask on Stack Overflow? A comparison of automated approaches to classify posts into question categories*. 2019.

[12] H. Zhang, S. Wang, T. Chen, A. Hassan and Y. Zou, *An Empirical Study of Obsolete Answers on Stack Overflow.* 2019, pp. 1-14.

[13] M. Saif, A. Medvedev, M. Medvedev and T. Atanasova, *Classification of online toxic comments using the logistic regression and neural networks models*. AIP Conference Proceedings, 2018, pp. 1-5.

[14] Chakrabarty, N., 2012. *A Machine Learning Approach To Comment Toxicity Classification*. West Bengal, India, pp.1-10.

[15]"Sampling Techniques", *Medium*, 2021. [Online]. Available: https://towardsdatascience.com/sampling-techniques-a4e34111d808.

[16] J. A. SÁEZ, M. GALAR and B. KRAWCZYK, *Addressing the Overlapping Data Problem in Classification Using the One-vs-One Decomposition Strategy*. 2019, pp. 83396-83411.

[17] M. S. Linneberg and S. Korsgaard, *Coding qualitative data: a synthesis guiding the novice*. Qualitative Research Journal, 2019, pp. 1-27.

[18]"Privileges - Comment everywhere", *Stack Overflow*, 2021. [Online]. Available: https://stackoverflow.com/help/privileges/comment#:~:text=What%20are%20comments%3F,they're%20gone%20for%20good.

[19]"How to Measure Quality when Training Machine Learning Models", *Labelbox Blog*. [Online].Available:https://labelbox.com/blog/how-to-measure-quality-when-training-machine-learning-models/.