Capstone Project 2- DISCORD NLP

Problem Statement

Many companies understand that providing a great customer experience is key for the success of a service or product. Customer Service is not only about fixing some billing questions or technical issues that customer face but also engaging their customer base to provide feedback on their experience.

Companies, such as Discord, have opened public feedback forums that customers can leverage to send improvement suggestions for Discord's API for example. However, some feedback is not correctly classified within the available categories (API, Mobile, Other, etc) causing to be missed by Discord customer service team.

Project Goal

The goal of my project is to answer the following question: **Is it possible to classify submitted Feedback topics to their proper category given their text?**

The answer to this question will help Discord in continuing to provide top notch customer service and also provide insight on potentially leads to development of new features or improvement of existing ones that its product management team could use.

Dataset

The historical data that will use to answer this question was scrapped from Discord's Feedback site and covers Dec, 2018 to Feb 2019.

Solution Approach

1. Phase 1: Text Wrangling and Pre-Processing: Build the corpus vocabulary
2. Phase 2: Text parsing and EDA
3. Phase 3: Text representation & Feature Engineering
4. Phase 4: Select ML Supervised algorithms/Evaluate results
5. Phase 5: Present insights (pattern/trends)

Deliverables

1. Code Jupyter Notebooks (Cleaning/EDA/ML algorithm)
2. Report deck

Phase 1

The first phase on the project consisted of retrieving data from Discord Feedback site, reviewing it and deciding the steps needed to transform text data into numerical data so selected models will be able to classify any new feedback topic to the correct category.

Dataset

From Discord Feedback site, I scraped feedback topics submitted by users from Dec 2018 to Feb 2019 by using Beautiful Soup. Stored the data in a .cvs format after removing "html' tags such using available classes from Beautiful Soup and Regex.

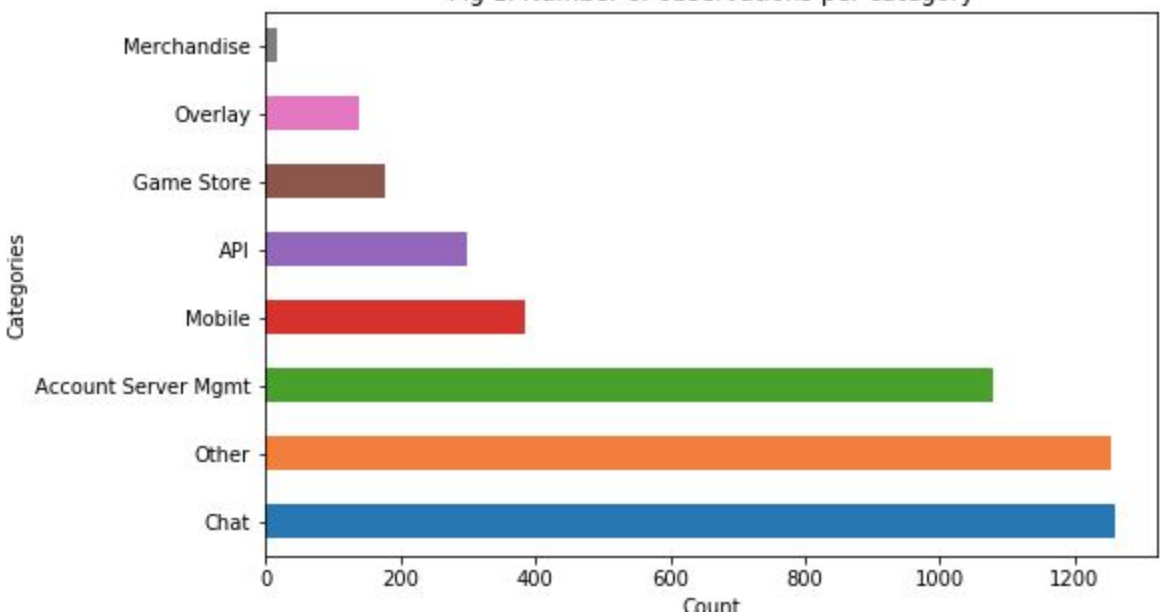
Data Review

Performed a preliminary analysis in Jupyter Notebook of the observations to understand the type of information contained in the feedback topic variable and votes variable. For example, topic variable included emojis, special character and non-english text. For the votes one, negative numbers were allowed. The data types in the dataset were integer and float.

The dataset contained over 4,500 observations that were assigned only to 1 of the 8 available categories. See table and figure 1 below:

```
Chat                  1260
Other                 1253
Account Server Mgmt   1078
Mobile                 386
API                    300
Game Store             177
Overlay                140
Merchandise             19
Name: categories, dtype: int64
```

Fig 1. Number of observations per category



Notice that given the small number of observations for some categories, a potential risk is that more "noise" might be created in the overall model. One way to confirm that this risk is now an issue is checking recall/precision scores obtained via the classification report.

Data Wrangling

To continue with the transformation process of the data, I focused my efforts first in topic feedback variable that contains the information that I want to feed the model in text format. Pre-processed the text by removing all unwanted special characters, contractions, especially the remaining html tags and lemmaziting the words by defining Text Normalized functions that returned a Normalized corpus.

Afterwards, started working on the numerical variable, Votes. This variable measures if other Discord users agreed with the suggested feedback. If a topic has 0 votes or negative votes, then users have a neutral sentiment or strongly disagree respectively. Users will show their agreement with a given topic by awarding positive votes.

Statistical functions that measured central tendencies were applied to the variable. From the table below, it is observed that the average votes a topic gets is 18.56, and its corresponding standard deviation is 355. The standard deviation value suggested that observations were spread out. This was expected given that a vote could take a negative value. The median for any topic is 4 votes. It can be noticed that users strongly agreed with a particular suggestion by giving it a max value of almost 20,500 votes but disagreed on another one by giving it -53 votes (min value).

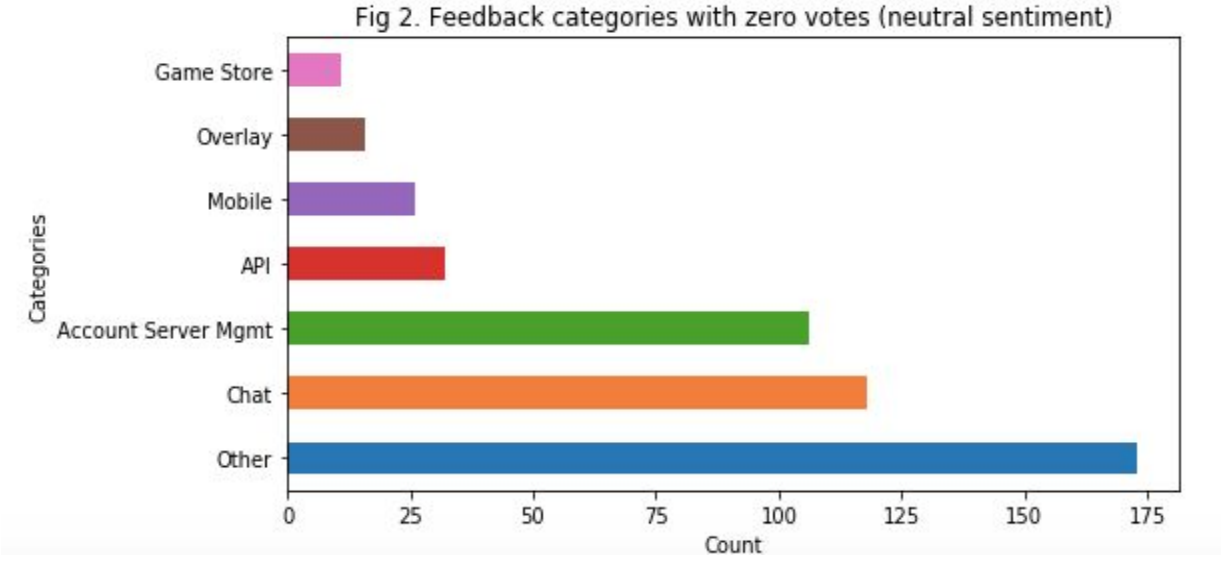
```
count     4613.000000
mean        18.558205
std        355.261912
min        -53.000000
25%          0.000000
50%          4.000000
75%         12.000000
max      20498.000000
Name: votes, dtype: float64
```

Digging a little further, the feedback topic that was voted the least popular with -53 votes was:

```
'discord should add so if you have nitro you can change your discord tag to c4 like csgos or just normal emojis but t
hen also have a second discord tag that you can use when someone is trying to add you'
```

Further analysis showed that feedback topics that users had neutral sentiments were distributed in 7 categories. Other was the category with highest number of topics with zero votes.

Fig 2. Feedback categories with zero votes (neutral sentiment)



A new dataframe was created with the Normalized corpus as “Clean description” plus “votes” and “categories” features. In this new dataframe, observations with missing data (NaN) and duplicates were removed from it. Before making the decision of dropping missing observations, other options were considered such as identifying what potential values could be used to fill them. However, it was difficult to differentiate if these missing observations were due to languages other than English, such as Japanese, Korean or to users filling them with special characters such as @, !, ‘.

Results

From performing these steps, the dataset was enhanced by:

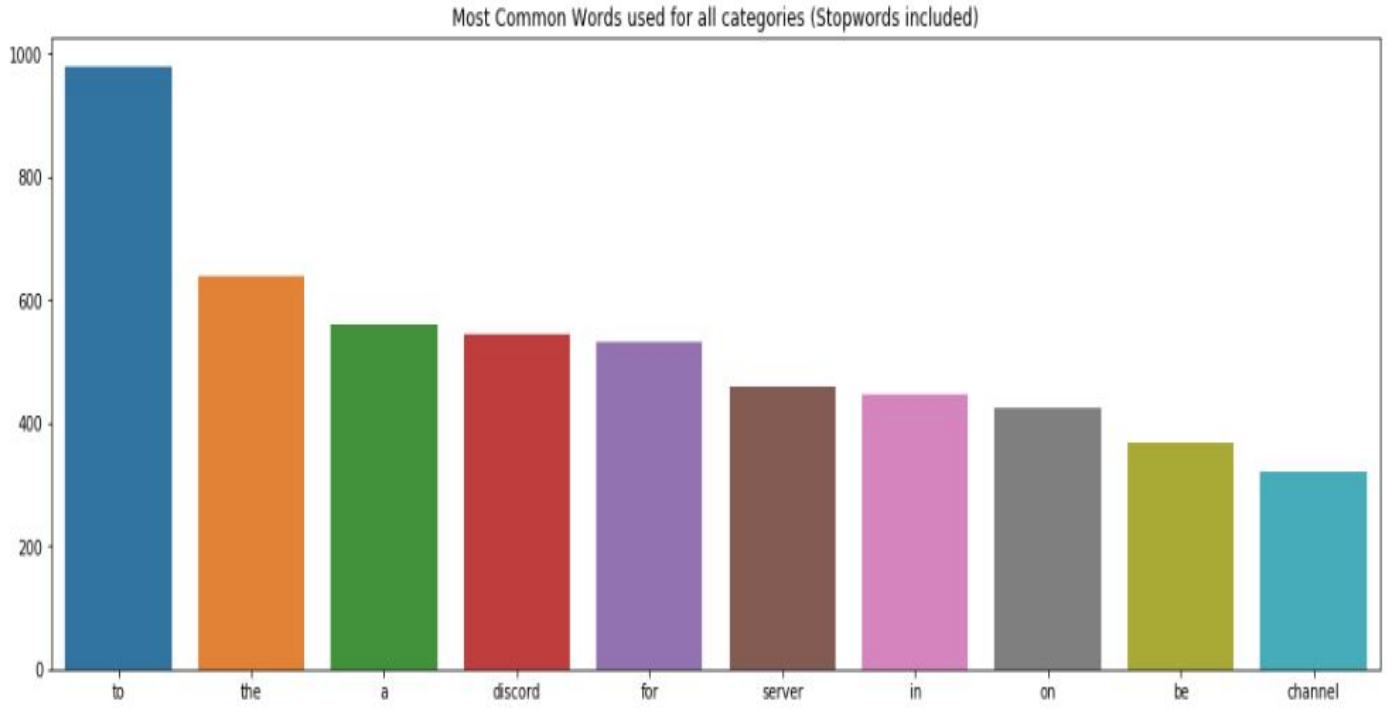
- Removing variables that had missing observations.
- Transforming the text variable to a Normalized Corpus.
- Keeping numerical values within the text observations.

Phase 2

Next, was to perform EDA on the text variable, “Clean description”, with the goal of testing the impact that removing Stopwords will have on it. To test the impact, the most common words used for all categories were found.

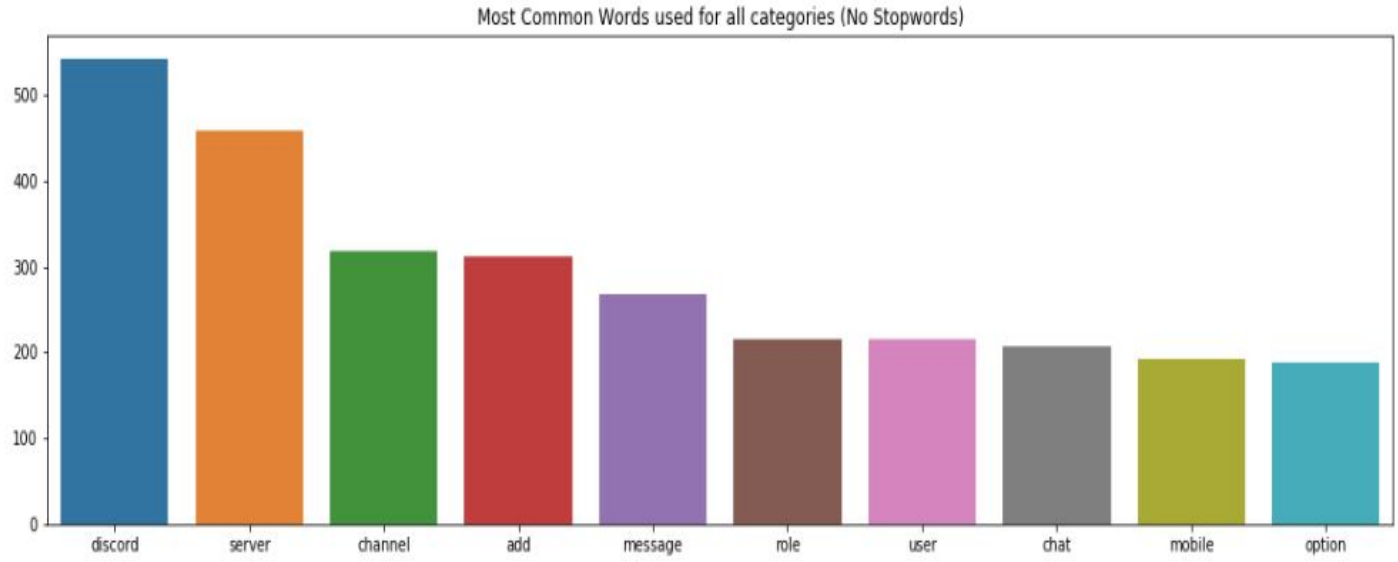
When stopwords were not removed, 7 out of 10 the most common words were stopwords (as shown in the graph below).

Fig. 3. Most common words across Disord Categories with stopwords



A function was created to tokenize and remove stopwords from text data. After function was applied, the clean text was plot in the graph below. Not surprisingly, Discord was the most used word by users. Other words that made it to the top had to do with “add”, “option”, “message”, “server”, “chat”, “mobile”. These words provided some clues on what users frequently submit feedback on. For example, such as suggesting to add an option that will impact Discord’s server infrastructure, chat or mobile features.

Fig. 4. Most common words across Disord Categories with no stopwords



Phase 3

The main dataset, containing the clean corpus (stopwords included), votes and categories variables, was partitioned into 2 sets: train and test. Text representation & Feature Engineering steps were performed on the split sets.

Two scenarios were considered and tested with the goal of obtaining the best performance out of the supervised learning models used.

Scenario 1: Removing the stopwords from the train/test datasets

Once the stopwords were removed from the train and test dataset, again most frequent words were identified and plotted. Not surprisingly, both plots had Discord as the most used word by users.

The next step was to transform the clean text from the train dataset into their vectorial representations aka Feature Engineering. The 2 models used were the Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).

The output of the Bag of Words model was a sparse matrix with a shape of 3032, 10872. This means the model created a vocabulary of 10872 unique words or features to represent the 3032 text observations! The next step was to convert the sparse matrix into a dataframe and then add the votes and categories variables.

Categories column was extracted from the newly created dataframe out of “train dataset”.‘ In this manner, Features and Label variables were ready for modeling. The ‘test dataset’ went through the same steps as described above.

The other model used was TF-IDF (default settings) and results were similar to the BoW model, meaning the same number of unique words, 10872, were obtained to represent the text data for both test/train datasets. Following the above steps, Label and Features variables were created.

Model	Features (Train)	Label (Train)	Features (Test)	Label (Test)
BoW	(3032, 10873)	(3032,)	(1494, 10873)	(1494,)
TI-IDF	(3032, 10873)	(3032,)	(1494, 10873)	(1494,)

Scenario 2: Not removing the stopwords from the train/test datasets

The same steps as in scenario 1 were followed and resulted in having the corresponding Label, Features variables from train/test datasets ready for modelling.

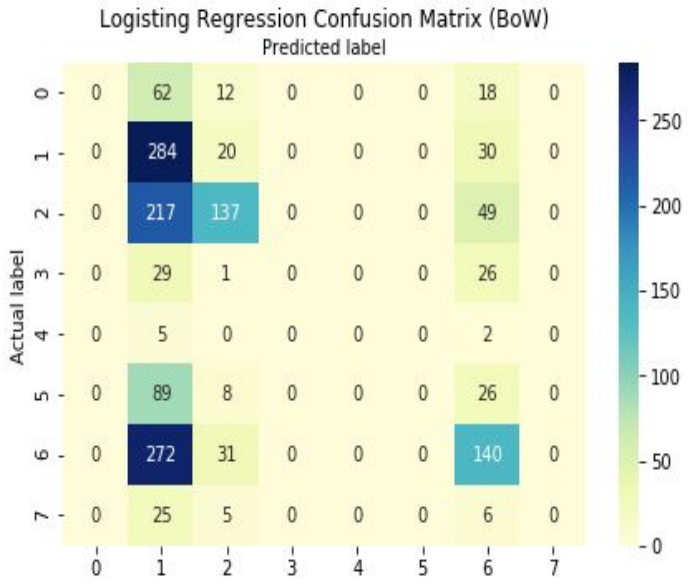
Model	Features (Train)	Label (Train)	Label (Train)	Label (Test)
BoW	(3032, 13003)	(3032,)	(1494, 13003)	(1494,)
TI-IDF	(3032, 13003)	(3032,)	(1494, 13003)	(1494,)

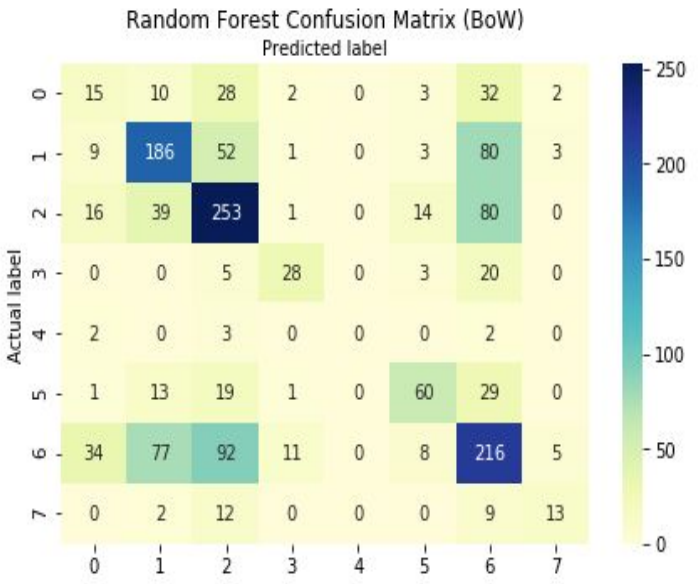
Phase 4

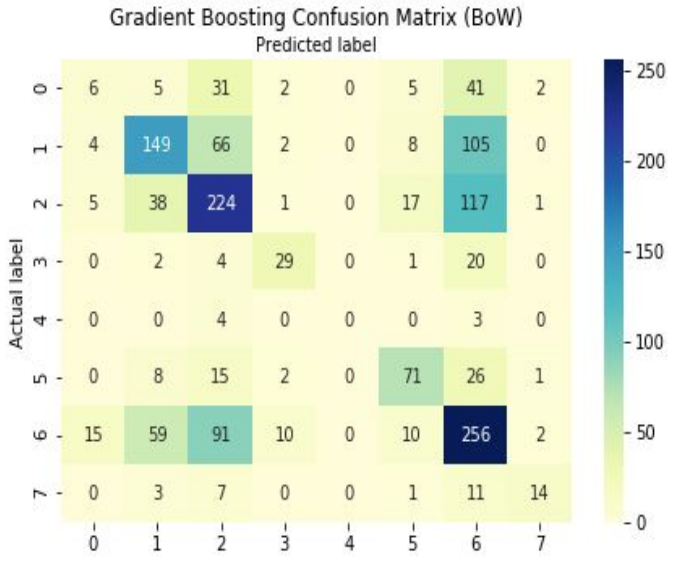
Now that the train and test “Features” plus “Label’ variables are ready to go, is time to model using the selected supervised models: Logistic Regression, Random Forest, Gradient Boosting. Final step will be to evaluate their performance using the available tools such as confusion matrix and classification report.

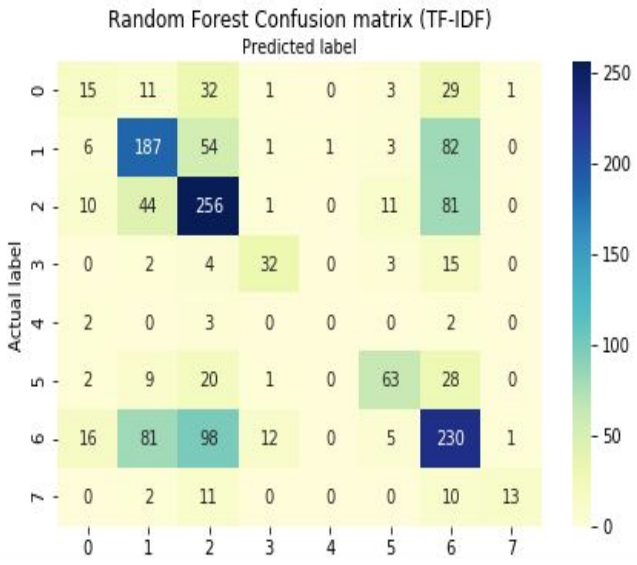
Scenario 1: Model Results (stopwords removed)

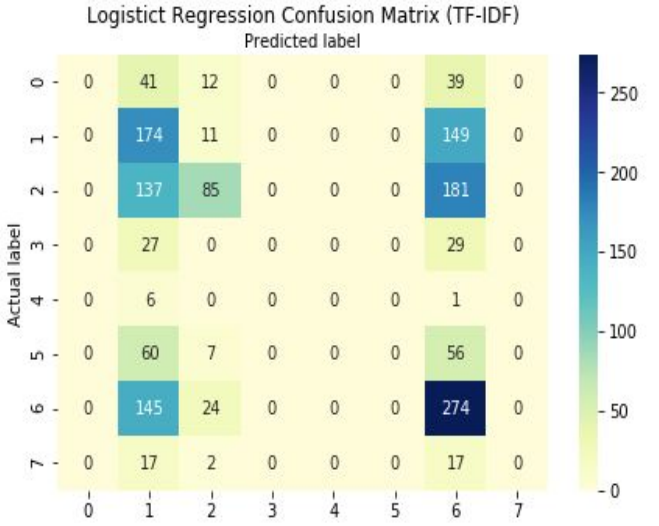
Fig. 5. Category **0**: API, **1**: Account Server Management, **2**: Chat, **3**: Game Store, **4**: Merchandise, **5**: Mobile, **6**: Other, **7**: Overlay

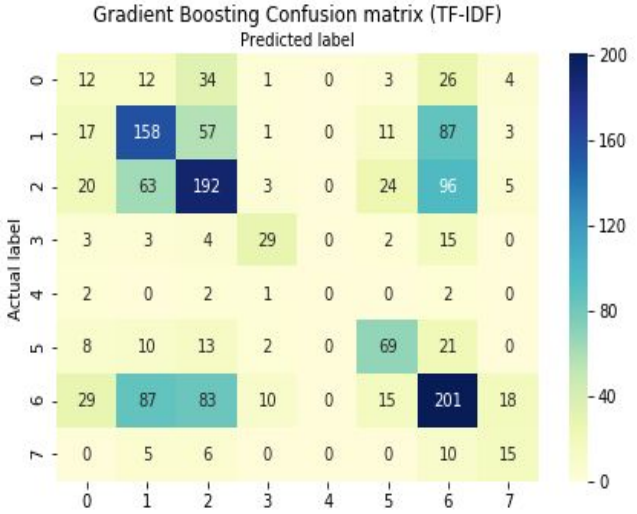








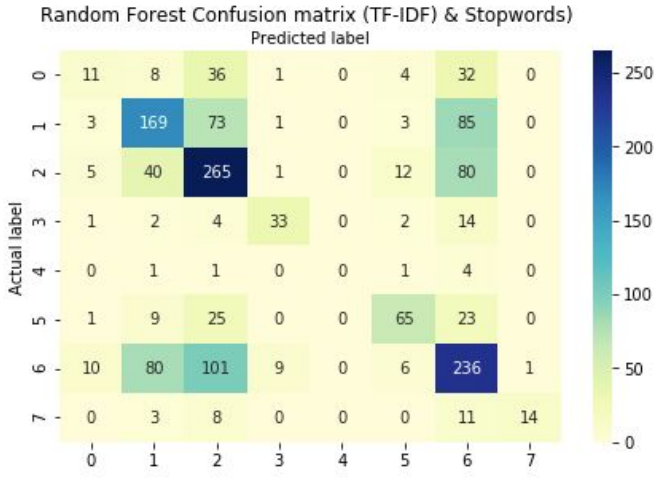


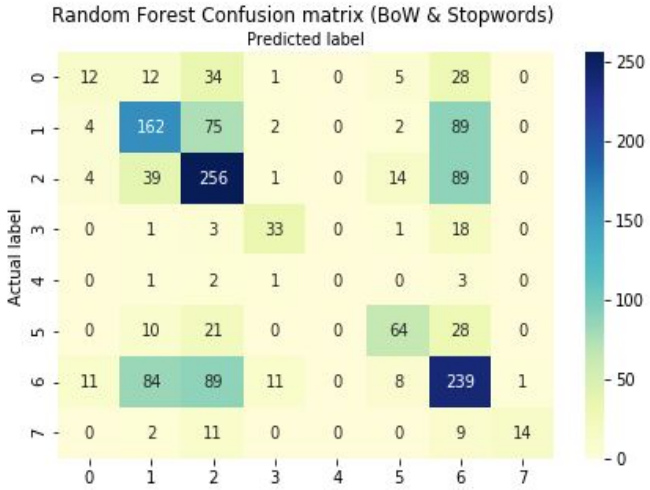


Scenario 2: Model Results (stopwords included)

Given that Random Forest model gave better results than the other models, only this model was used during Scenario 2.

Fig. 6. Category **0**: API, **1**: Account Server Management, **2**: Chat, **3**: Game Store, **4**: Merchandise, **5**: Mobile, **6**: Other, **7**: Overlay





Phase 5

For this particular project, I am assuming that Discord is looking for a holistic picture on how the topics are being classified by the models before selecting one. With that in mind, F1-score will be used for evaluation purposes.

The classification report for to determine which model performs the best for all classes, showed that Random Forest (TF-IDF) for Scenario 1 worked the best with an average
F1-score of **.53.**

	precision	recall	f1-score	support
API	0.29	0.16	0.21	92
Account Server Mgmt	0.56	0.56	0.56	334
Chat	0.54	0.64	0.58	403
Game Store	0.67	0.57	0.62	56
Merchandise	0.00	0.00	0.00	7
Mobile	0.72	0.51	0.60	123
Other	0.48	0.52	0.50	443
Overlay	0.87	0.36	0.51	36
avg / total	0.53	0.53	0.53	1494

For Scenario 2, the best performing was also Random Forest (TF-IDF) with average F1-score of **.52.**

	precision	recall	f1-score	support
API	0.35	0.12	0.18	92
Account Server Mgmt	0.54	0.51	0.52	334
Chat	0.52	0.66	0.58	403
Game Store	0.73	0.59	0.65	56
Merchandise	0.00	0.00	0.00	7
Mobile	0.70	0.53	0.60	123
Other	0.49	0.53	0.51	443
Overlay	0.93	0.39	0.55	36
avg / total	0.53	0.53	0.52	1494

However before calling it done, let's review how the model performed for each class or category. This analysis shows another picture. Neither of the models was able to classify correctly the topics for Merchandise category. Indicating this one is creating noise and the model is not learning

For both scenarios, Random Forest model obtained the highest F1 scores when classifying topics to Game Store and Mobile categories. In both scenarios, the winning models had trouble when classifying topics to the API category as observed from their respective F1 scores but clearly not having stopwords was key.

An interesting category to pay closer attention is 'Other'. In theory, the user will log feedback that it is not related to the available 7 categories. However, that was not the case after quickly reviewing submitted content. It became clear that users were including feedback that should be part of these 7 categories. For example, the confusion matrix for Random Forest (TF-IDF & without stopwords) showed that 179 "Other" observations should be classified under Account & Server Management and Chat. Something similar is observed for the Random Forest (TF-IDF & Stopwords) model.

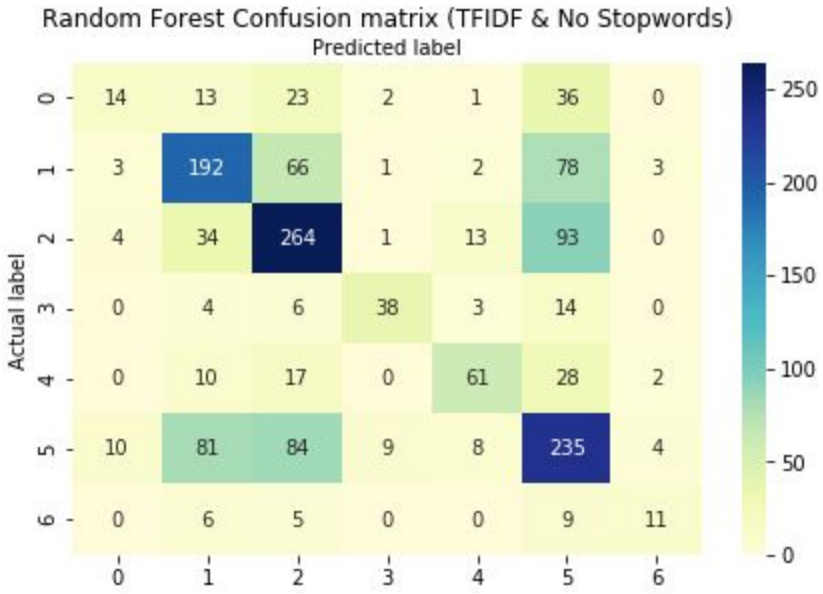
Round 2: Improving performance for each class

It was decided to remove Merchandise observations from the dataset to test the assumption that better classifications results would be obtained from the remaining categories.. After extracting them, the labels/features (table below) were obtained for the train/test set.

Model	Features (Train)	Label (Train)	Label (Train)	Label (Test)
TI-IDF	(3020, 10858)	(3020,)	(1488, 10858)	(1488,)

Again stopwords were removed and Random Forest model showed the best performance as observed in the Confusion matrix and classification report for the 7 categories.

Fig. 7. Category **0**: API, **1**: Account Server Management, **2**: Chat, **3**: Game Store, **4**: Merchandise, **5**: Mobile, **6**: Other, **7**: Overlay



	precision	recall	f1-score	support
API	0.45	0.16	0.23	89
Account Server Mgmt	0.56	0.56	0.56	345
Chat	0.57	0.65	0.60	409
Game Store	0.75	0.58	0.66	65
Mobile	0.69	0.52	0.59	118
Other	0.48	0.55	0.51	431
Overlay	0.55	0.35	0.43	31
avg / total	0.55	0.55	0.54	1488

Overall, the average F1 scores went up for all categories. Game Store improved the most, followed by API. Showing that Random Forest model is doing better in learning to differentiate valuable signals from the noise. Although F1-score from “Other” category did not go down as expected, at least it remained the same and business should confirm if the value obtained is when “Other” topics are not being classified correctly/

Hyperparameter Tuning

Another way to improve model performance was to try out different values for some hyperparameters. For this scenario, used the same dataset with 7 categories and compared Random Forest default model to one with 150 trees using entropy as the criterion. Classification report for the Random Forest model (150) indicated that the default model of n=100 trees performed better with F1-Score of .53..

	precision	recall	f1-score	support
API	0.24	0.14	0.18	92
Account Server Mgmt	0.53	0.51	0.52	334
Chat	0.54	0.64	0.58	403
Game Store	0.70	0.59	0.64	56
Merchandise	0.00	0.00	0.00	7
Mobile	0.69	0.48	0.56	123
Other	0.48	0.53	0.50	443
Overlay	0.87	0.36	0.51	36
avg / total	0.52	0.52	0.52	1494

Using Stratify

Another approach used to improve the model performance was to use stratify when splitting the dataset (containing 8 categories) so the proportion of labels produced in train/test split will be the same as in the input dataset.

Since removing stopwords was adding more value to the model, they were removed. I went ahead and used Bag of Words (BoW) and TF-idf techniques to transform the text into its corresponding vector representation. Applied again the supervised models of: Logistic Regression, Random Forest, Gradient Boosting,

Logistic Regression (BoW) was the model that showed the most improvement when using stratify.

Classification Report (Stratify)

	precision	recall	f1-score	support
API	0.00	0.00	0.00	98
Account Server Mgmt	0.60	0.50	0.54	344
Chat	0.50	0.66	0.57	412
Game Store	0.70	0.12	0.21	58
Merchandise	0.00	0.00	0.00	6
Mobile	0.62	0.42	0.50	126
Other	0.41	0.58	0.48	405
Overlay	0.00	0.00	0.00	45
avg / total	0.47	0.49	0.46	1494

Classification Report (Original)

	precision	recall	f1-score	support
API	0.00	0.00	0.00	92
Account Server Mgmt	0.29	0.85	0.43	334
Chat	0.64	0.34	0.44	403
Game Store	0.00	0.00	0.00	56
Merchandise	0.00	0.00	0.00	7
Mobile	0.00	0.00	0.00	123
Other	0.47	0.32	0.38	443
Overlay	0.00	0.00	0.00	36
avg / total	0.38	0.38	0.33	1494

It is observed that recall improved the most for Mobile and Chat categories, which as clear indication that having that uniform distribution of keywords in the train dataset was critical. Even the model learned to identify observations that belonged to Game store which was not the case for the original model.

Such drastic increase was not observed for the winning model of Random Forest (Tf-idf) and actually the F1-Score decreased to .50. This could be due to the fact that TF-Idf already normalized the words and thus reduced the importance of words that could have improved the classifications.

Data Science Results/Insights

- Overall Random Forest model worked the best in the different scenarios tested.
- Given the noise in our observations, removing stopwords was a critical task.
- To balance these dataset, an option to consider is the downsampling of the categories with the highest number of observations.
- Models could not learn to classify observations that belonged to Merchandise due to its low number of observations. Getting more observations was not possible since users were not submitting them..The recommendation is not to include these observations in the next model iteration.
- For the observation under "Other" category, the model learned and suggested the potential categories for the observations that were not truly other.
- Expand the stopwords list to include words like Discord, other. Los.

Business Insights

- Customer success team now has a model that is able to automate the Classification process and filter records to their corresponding categories at least 50% of the time. This is a big improvement from not having one.
    - Still agents are needed to provide their feedback and use that information to improve our model.
- Customer success should compare the words identified by the model as the top contributors against existing tag list used to filter/route submissions and decide if any of them should be included.
- Customer Success Leadership needs to provide guidance on the baseline performance metrics that each category should achieve after using the model. My recommendation is to focus on the recall score since it provides better performance coverage for all classes and identifies the relevant ones.