

Computer Science Capstone

Toxic Comment Analyzer

Stephen Hall

Student ID: 001183455

Western Governors University

Table of Contents

Table of Contents	2
Section A – Project Proposal	4
Problem	4
Application Benefits	4
Application Outline	4
Data Used	5
Objectives	5
Hypothesis	5
Project Methodology	5
Funding Requirements	6
Stakeholder Impact	6
Ethical and Legal Considerations	6
Expertise	6
Section B – Executive Summary	7
Problem Statement	7
Customers	7
Existing System Analysis	7
Available Data	7
Methodology	8
REQUIREMENTS ANALYSIS	8
SYSTEM DESIGN	8
IMPLEMENTATION	8
TESTING	8
DEPLOYMENT	9
MAINTENANCE	9
Deliverables	9
Implementation Plan	9

Validation Methods	9
Environments	10
Timeline	10
Section C – Attributes	10
Descriptive Method	10
Non-descriptive Method	10
Decision Support	11
Data Preparation, Exploration and Visualization	14
Interactive Queries	25
Machine Learning	26
Accuracy Evaluation	27
Security	27
Monitor and Maintenance	28
Dashboard	28
Section D – Documentation	31
Business Requirements	31
Code for Predictive Model	34
Hypothesis Assessment	39
Visualization and Data Explorations	41
Accuracy Assessment	41
Application Testing	41
Source Code and Executable Files	43
Quick Start Guide	43
Section E: References	45

Section A – Project Proposal

Problem

Toxic comments and cyberbullying are a major problem for any social media platform or online form. In almost every social media platform or online form, users are allowed to remain anonymous, identified by only a username or email. Almost no moderation exists, and when social media is moderated, it is always after the comment has been posted and the damage is already done. In an effort to eliminate cyberbullying and toxic comments, a predictive tool can be made to analyze hundreds of thousands of existing comments, examine them in real time, and prevent them from being posted.

Application Benefits

The application is to be used to identify toxic comments in real time that would be posted on social media platforms or online forms. Its use is encouraged for social media platforms and online forms in the prevention of cyberbullying and toxic comment prevention. the application ultimately benefits in helping to create a positive online experience while helping to prevent cyberbullying.

Application Outline

The application utilizes Python's matplotlib, and seaborn libraries to generate data visualization charts and images using the test data in real time. The Scikit-Learn library is largely responsible for the model in Corpus cleaning of data and performing the Cramer's V statistic, Feature Engineering, and Feature Importance for categorical-categorical association on the comment data and to create a baseline model then uses logistic regression to predict the probabilities if a comment is toxic or not.

Additionally, a React.js application is also provided for a live use case example using the tensorflow.js library. A Social media company may want to implement this application as middleware to prevent toxic comments in real time from ever being posted.

Data Used

The data used to build the product model will be mined from the Toxic Comment Classification Challenge (Jigsaw 2017).

We expect the following variables from the prototype to also exist in the product's model.

Toxic	Binary
Severe_Toxic	Binary
Obscene	Binary
Threat	Binary
Insult	Binary
Identity_Hate	Binary

Additional data may include IP address, comment, and username.

Objectives

This project's primary objective is to identify and stop toxic comments in online forms and social media platforms. The ultimate objective is to eliminate cyberbullying, spammer, insulting comments online.

Hypothesis

A predictive model can identify toxic comments and levels of toxicity in comments posted online.

Project Methodology

The product shall be developed and supported using the agile methodology because the requirements are well defined and the speed of implementation is important. Social media platforms and online forms require easy of use and vendor specific details in order for adoption, so being able to adapt to the customers needs in the implementation makes this project perfect for the agile model.

1. Requirements Analysis – ensure requirements are feasible, testable, and realistic

2. System Design – ensure the system is stable and meets the requirements for each customer
3. Implementation – ensure application is functional, able to receive regular updates, meets the customers needs, and has a pipeline developed and in place for reinforcement learning
4. Testing – integrate the application into the customers existing systems and test for faults, failures, and any incompatibilities in the software.
5. Deployment – install the product onto the customers cloud server environment
6. Maintenance – continue to feed the model new trained and untrained data to evaluation and ensure its continued efficacy, relevancy, and accuracy.

Funding Requirements

Funding for the project requires an upfront cost of \$42,500 for development and roughly \$5,500/year in cloud services and routine maintenance. The costs may vary depending on the amount of use for the cloud services so a best estimate is given.

Stakeholder Impact

Reddit is one of the oldest social media platforms online today. With hundreds of thousands of subreddits and millions of comments being posted each day, cyberbullying and toxic comments run rampant. There are just not enough man hours to moderate such an influx of data each day. Integrating the toxic classifier into the posting system and allowing moderators to set the level of allowed toxicity would allow for communities with a greater sense of safety and a decrease in cyberbullying and online threats. Using the application would allow users to feel safer online and engage more in the Reddit communities allowing for growth of the user base and overall increase in profits.

Ethical and Legal Considerations

The dataset is anonymous comment data with no personally identifiable information which is not sensitive nor protected. The model does not save or store predictive results. The data cannot be reverse engineered to get personal information and meets all the privacy laws and guidelines including the GDPR (GDPR 2019).

Expertise

The developer has 10 years of software engineering experience including 8 years of web and mobile

app development and 5 years in advanced artificial intelligence and machine learning specific applications.

Section B – Executive Summary

Problem Statement

Cyberbullying has been on the rise in recent years. According to huffpost.com, 33.8% of teens ages 12-17 have been cyberbullied (huffpost.com 2017). These numbers are continuing to climb each year. With comments on online forms and social media being the biggest contributors, a predictive analysis tool can be used to prevent a large majority of these toxic comments.

Customers

The product is meant to be used as a middleware application inside of social media platforms and online forms. The information required to use the model can be administered by the programmers or network administrators easily as it does not require any changes to the existing program structure. This makes the product perfectly suited for social media platforms like Facebook, Instagram, Reddit, etc. All that would be required is to take the comment posted by a user and send it to the middleware model. The middleware model will return its classification and percentage score for the likelihood of being toxic. This intended behavior can be seen in the react application. The model is suited to run on either a subscription based or by call pricing model based on the amount of data being submitted from the customer.

Existing System Analysis

There are currently no preventive measures outside of manual human intervention in play for stopping and preventing toxic comments and cyberbullying on social media platforms or online forms. Some social media sites have banned certain words or phrases from being posted, but nothing is in place to fight cyber bullying of toxic comments online.

Available Data

The required data is published on the Kaggle - Toxic Comment Classification Challenge website. This can be found in the following hyperlink.

<https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data>

This dataset contains over 100,000 comments posted by users on various online forms and social media platforms. The data was then crowdsourced to be tagged by its level of toxicity for the training data.

The analyst will further filter results that pertain to behaviors, risk factors, and toxicity levels to compile the raw dataset. This dataset will then be cleaned for use in modeling using the methods similar to those used to build the prototype.

Methodology

This product will employ the agile methodology for the software development process.

REQUIREMENTS ANALYSIS

The system and software requirements will be broken down into tasks. These tasks will then be assigned into sprints each day to ensure the requirements are feasible, testable and completed.

SYSTEM DESIGN

The team will design the system in such a way that more or less hardware and traffic load resources can be spun up in the Amazon S3 environment to adapt to customers needs for each implementation. Using this information, the document and designs will be updated in the requirements document and built to specification.

IMPLEMENTATION

Per the system design, the product will be developed in such a way that is can dynamically spin off more or less resources or load balancers to fit all customers needs. This will require the following components to be developed: the dashboard, model, dynamic AWS S3 servers, Load balancers, and web deployment. Unit testing will ensure that individual components are functional and meet all the requirements.

Additionally, as new data is scrapped to support the model, an ongoing learning process will be implemented into the model to continue to improve its accuracy and efficiency. The model will be especially interested in comments who were flagged as high risk but was not found to be a toxic comment to reduce the false positive rate over time.

TESTING

As each component is unit tested, it will then be integrated into the system and the system will be tested for faults and failures.

DEPLOYMENT

The product will first be installed on an AWS S3 server to serve as a repository for the docked model. The product will then be deployed with user customized dashboards to be accessed over the internet to various locations throughout the world.

MAINTENANCE

Monthly maintenance procedures will be performed on the new data gathered to support reinforcement learning of the model. Modifications and new feature can also be requested at this point to be developed and implemented for updating the system..

Deliverables

A central location will serve as the hub for all connected customers created on an AWS S3 server.

The product will be accessible for customers through their own personal dashboards. The dashboard will contain all necessary information and code snippets for customers to quickly and easily integrate the middleware into their existing platforms requiring very little down time or training.

A database will be built and maintained for all the comment classification data that will be updated with each monthly recalculation of the reinforcement learning. This will allow the model to quickly adapt and accommodate changes in classifications of comments over time.

Implementation Plan

The implementation plan can be laid out as follows.

1. Perform data mining to acquire a raw dataset of new comment data.
2. Utilize descriptive techniques to achieve variable reduction
3. Utilize non-descriptive techniques to build the predictive model
4. Create a dashboard to capture data visualization and predictive model calculations
5. Integrate the predictive model into the middleware
6. Install the model onto an AWS S3 Server to serve as a hub
7. Perform acceptance testing and modifications
8. Perform monthly reinforcement learning and recalculations with new data.

Validation Methods

The software is validated by accuracy reports provided by customers that use the model. This will help to verify that the accuracy is within range of the original model and help in the reinforcement learning. Each model's sensitivity, specificity, false positive rate, and false

negative rate should be closely monitored each month to ensure that the model continues to learn.

Environments

Cloud Server	AWS	\$55/month
Development Cost (analytics)	90 hours - \$250/hour	\$22,500
Development Cost (software)	80 hours - \$250/hour	\$20,000
Maintenance	\$200/hour - 30 hours/year	\$6,000/year
Up front cost:	\$42,500	
Estimated ongoing cost:	\$ 5,500 per year ~ \$460 per month	

Timeline

<u>Phase</u>	<u>Begin</u>	<u>End</u>	<u>Duration</u>
Requirements Analysis	12/01/2019	12/15/2019	15 days
System Design	12/15/2019	12/30/2019	15 days
Implementation	01/01/2020	01/31/2020	31 days
Testing	02/01/2020	02/28/2020	28 days
Deployment	03/01/2020	03/07/2020	7 days

Section C – Attributes

Descriptive Method

The model was built using Cramer's V statistic, feature engineering, and feature importance to determine which predictor variables were used by the model.

Non-descriptive Method

The model used a baseline model with logistic regression to predict the likelihood that a comment is classified as toxic.

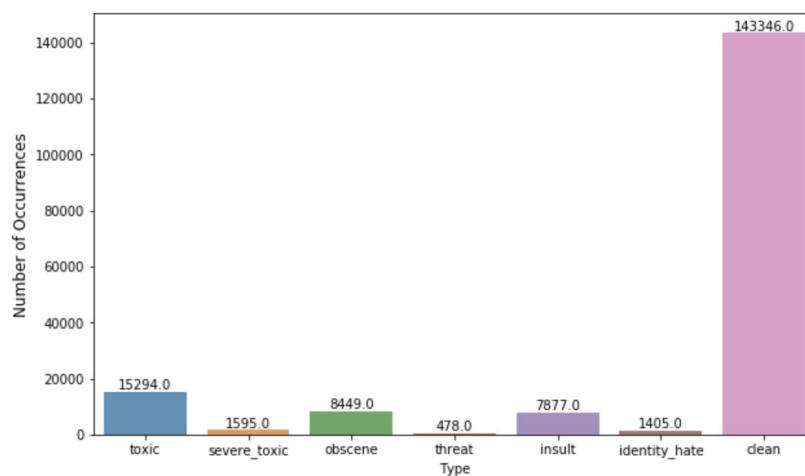
Decision Support

The product first gathers the trained data and organizes it by tags.

Loading the trained data:

Total Comments = 159571
Total Clean Comments = 143346
Total Tags = 35098

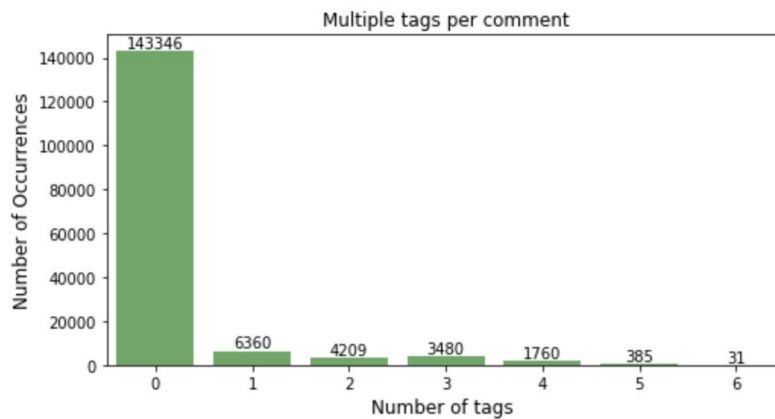
Show totals for the trained data



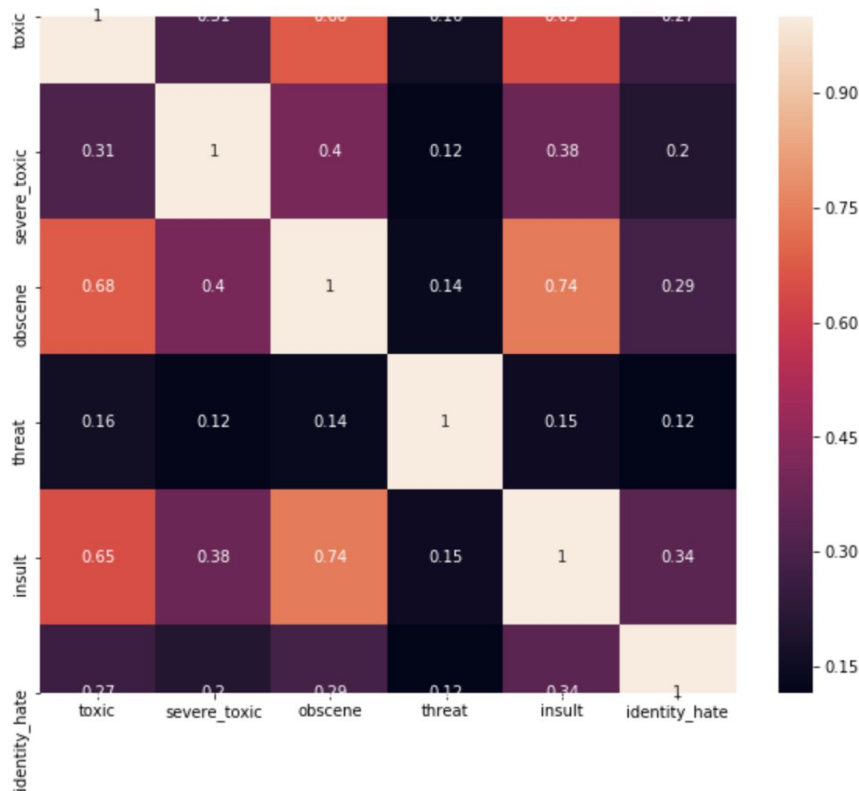
After that, the data is then grouped calculating how much of the data contains multiple tags per comment.

Comments with Multitpal Tags:

Let's check how many comments have multiple tags.



This data is then used to create a Correlation Plot identify how often the comments are classified with more than one tag, i.e as Clean, Toxic, Severe Toxic, Obscene, Threat, Insult, and Identify Hate.



The desired outcome is, of course, to accurately predict as many comments as possible into one of these categories. Using the correlation plot a crosstab is performed against the toxic and severe toxic for the remaining tags as it is not possible to create a visualization between 6 variables. Then a confusion matrix is created against the Camer's V statistics between toxic and severe toxic.

	severe_toxic		obscene		threat		insult		identity_hate	
severe_toxic	0	1	0	1	0	1	0	1	0	1
toxic										
0	144277	0	143754	523	144248	29	143744	533	144174	103
1	13699	1595	7368	7926	14845	449	7950	7344	13992	1302

[Edit Metadata](#)

The table above is a reproduction of a Crosstab/confusion matrix of the Toxic comments.

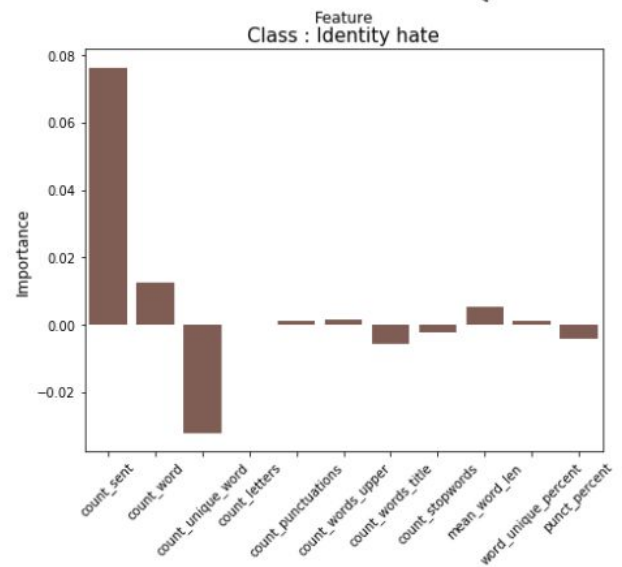
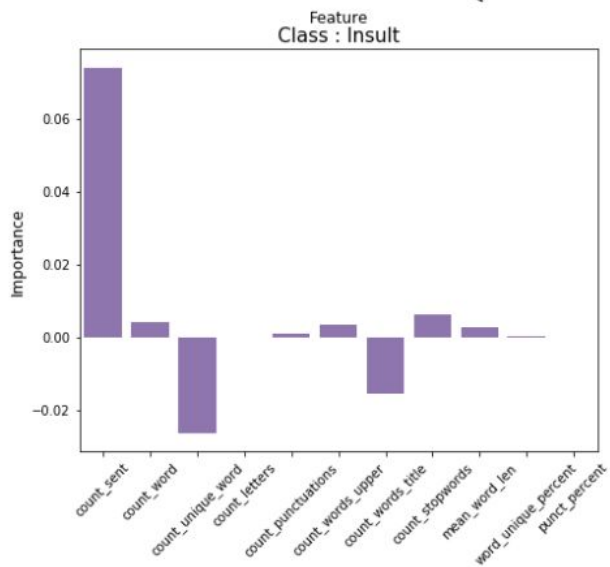
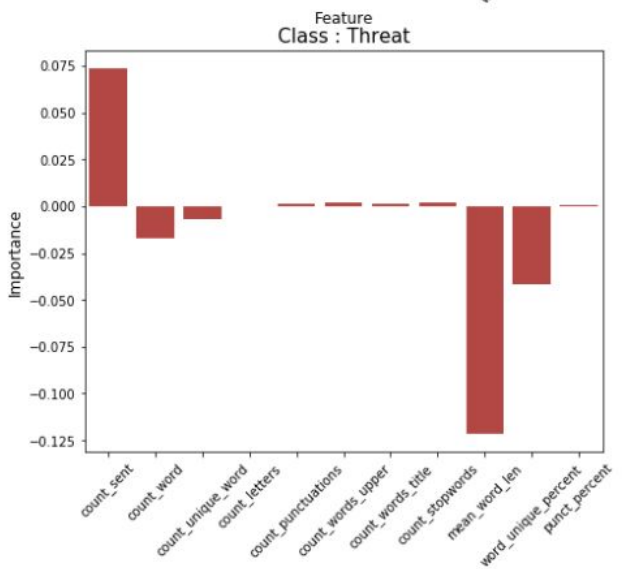
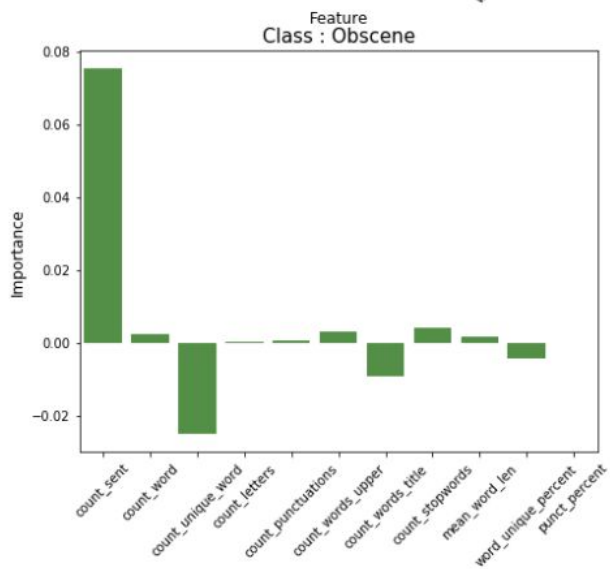
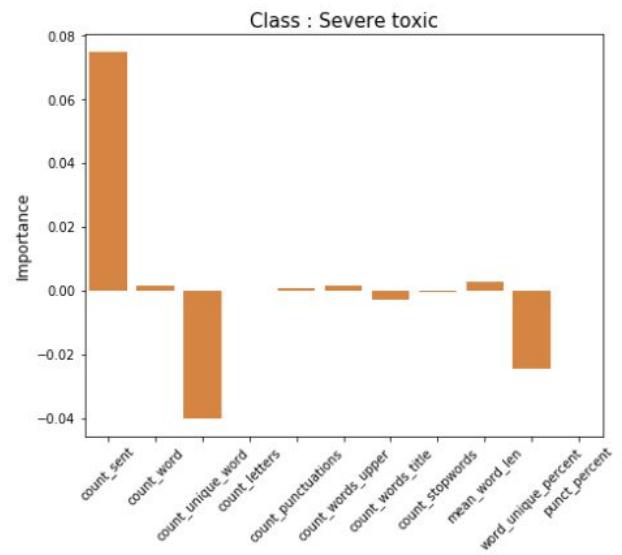
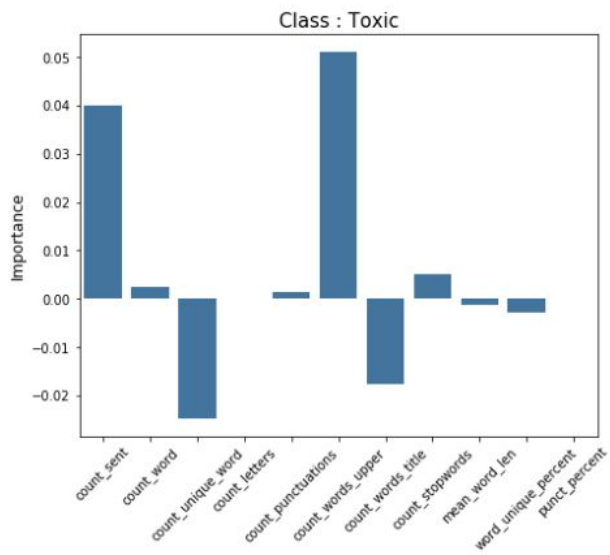
Confusion matrix for toxic and severe toxic:

```
severe_toxic    0    1
toxic
0              144277    0
1              13699  1595
```

The correlation between Toxic and Severe toxic using Camer's V statistic= 0.30850290540548614

Using the trained data as a base model, the testing data is then explored and classified with a variety of features explained in the next section to create the following classification data:

Feature importance for indirect features



The ability to classify the data based on the context in question results form this model as shown below:

Your comment	% of toxicity	✓ ⚠	Label
I'm only happy when it rains	1%	✓	Comment is clean
I want to watch the world burn	12%	⚠	Comment clasified as threat
I hope you die	82%	⚠	Comment clasified as insult

Notice that even though the second comment has only a 12% toxicity score, the model was still able to identify the comment as one that is toxic based on the context of the comment using Sentiment scoring.

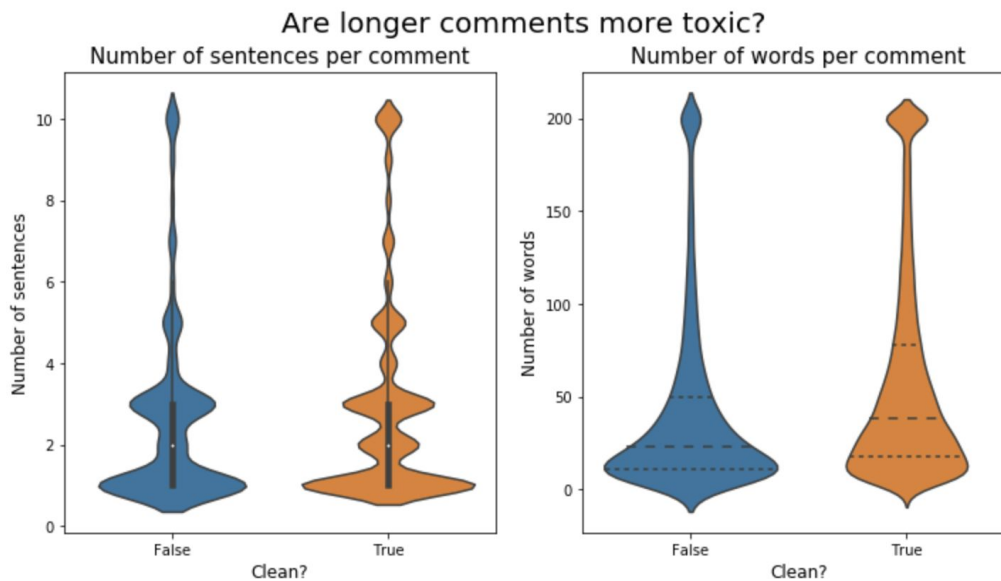
Data Preparation, Exploration and Visualization

Now that the trained data has been classified, the testing data is explored using the following Feature engineering techniques:

- Word frequency including:
 - Count features
 - Bigrams
 - Trigrams

- Vector distance mapping
- Sentiment scores
- count of sentences
- count of words
- count of unique words
- count of letters
- count of punctuations
- count of uppercase words/letters
- count of stop words
- Avg length of each word

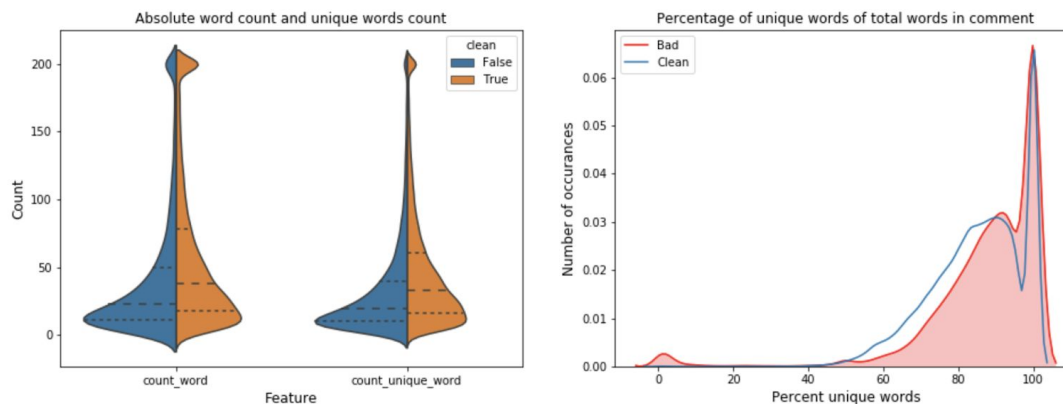
The testing data is used to determine is longer comments are more toxic.

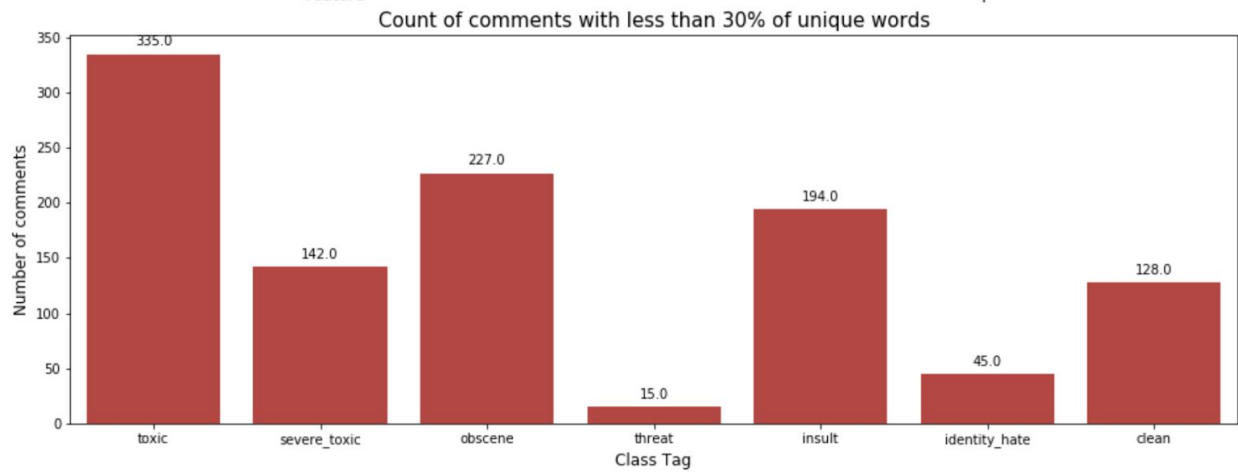


From the data it seems that longer comments do not seem to be a significant indicator of toxicity.

Chart desc: The inner markings show the percentiles while the width of the violin charts shows the volume of comments at that level.

Gathering Unique Words





The findings show that comments that contain less than 30% of unique words are more likely to be toxic than not. This data is used to strengthen the model.

In order to prevent data leakage, the leaky variables are identified so that they can be visualized and removed.

Cleaning Leaky Data

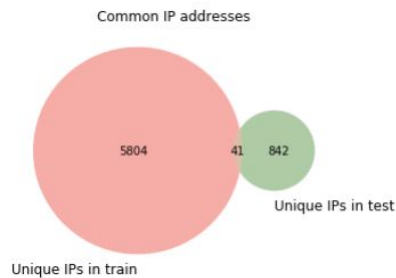
Caution: Including this data can lead to unexpected results, it will not make sense to add them into the final model, so the data will be cleaned.

Here we are creating our own custom count vectorizer to create count variables that match our regex condition

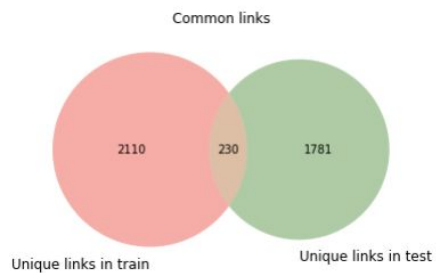
Leaky Usernames

```
[ 'destruction',  
  'diablo',  
  'diligent',  
  'dland',  
  'dlohcierekim',  
  'dodo',  
  'dominick',  
  'douglas',  
  'dpl',  
  'dr' ]
```

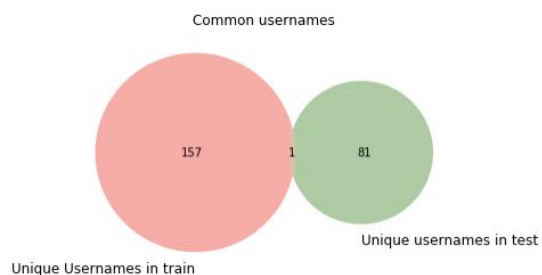
Leaky IPs Count



Leaky Links Count



Leaky Usernames Count



The leaky data between the training and testing data, as well as common contraction and other leaky is then removed using corpus cleaning.

A data dictionary is created to aid with this as shown below;

```
# Lookup dict of common contractions and modern spellings
lDict = {
    "aren't" : "are not",
    "can't" : "cannot",
    "couldn't" : "could not",
    "didn't" : "did not",
    "doesn't" : "does not",
    "don't" : "do not",
    "hadn't" : "had not",
    "hasn't" : "has not",
    "haven't" : "have not",
    "he'd" : "he would",
    "he'll" : "he will",
    "he's" : "he is",
    "i'd" : "I would",
    "i'd" : "I had",
    "i'll" : "I will",
    "i'm" : "I am",
    "isn't" : "is not",
    "it's" : "it is",
    "it'll" : "it will",
    "i've" : "I have",
    "let's" : "let us",
    "mightn't" : "might not",
    "mustn't" : "must not",
    "shan't" : "shall not",
    "she'd" : "she would",
    "she'll" : "she will",
    "she's" : "she is",
    "shouldn't" : "should not",
    "that's" : "that is",
    "there's" : "there is",
```

NOTE: the actual dictionary is much larger in the code.

The corpus cleaning method created to clean the data can be shown below with a before and after shot with some example data:

The above method has cleans the data like so:

```
def clean(comment):
    """
    This function receives comments and returns cleaned word-list
    """
    #Convert to lower case
    comment=comment.lower()
    #remove newlines
    comment=re.sub("\\n", "",comment)
    # remove leaky elements like ip & user
    comment=re.sub("\\d{1,3}\\d{1,3}\\d{1,3}\\d{1,3}", "",comment)
    #removing usernames
    comment=re.sub("\\[\\.\\*\\]", "",comment)

    #Split the sentences into words
    words=tokenizer.tokenize(comment)

    # (')aphostrophe replacement (ie) you're --> you are
    # ( basic dictionary lookup : master dictionary present in a hidden block of code)
    words=[lDict[word] if word in lDict else word for word in words]
    words=[lem.lemmatize(word, "v") for word in words]
    words = [w for w in words if not w in eng_stopwords]

    # returning cleaned data
    clean_sent=" ".join(words)
    return(clean_sent)
```

Example corpus data:

'\n\n NOTE If you read above, and follow the links, any reader can see that I cited correctly the links I added on this subject. Vidkun has added anotations to make them read as the oposite, but these links s how the "official" line taken by UGLE. I will not be trapped by any User into so-called 3RR, so he can peddle his POV. Strangly, ALL other "MASONS" are quiet, leaving 'me' to defend that factual truth on my own. "Thanks" Brethren. Sitting any blocking out if given... ''

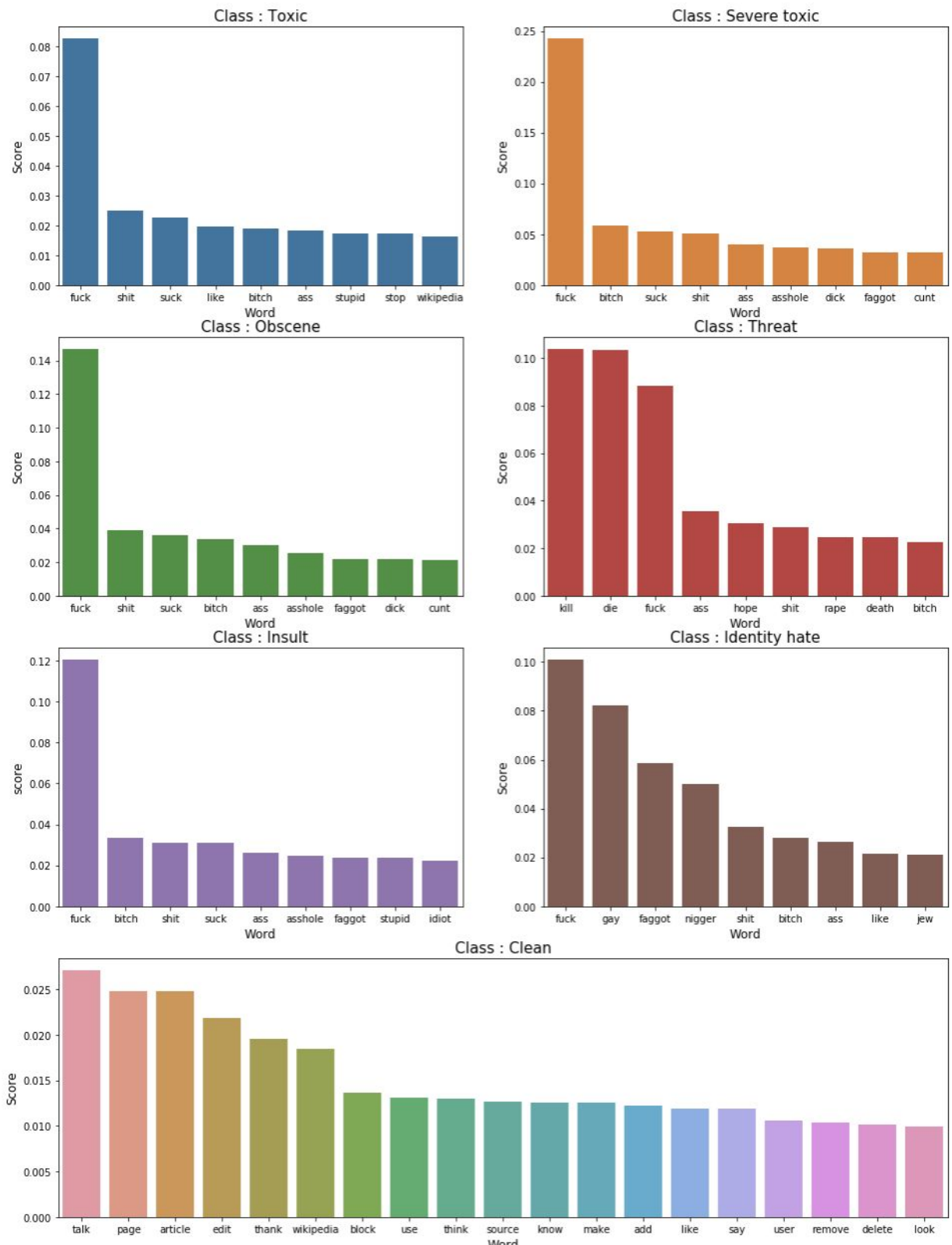
Cleaned corpus data:

'" note read , follow link , reader see cite correctly link add subject . vidkun add anotations make read oposite , link show " " official " " line take ugle . trap user so-called 3rr , peddle pov . strangly , " " masons " " quiet , leave ' ' ' defend factual truth . " " thank " " brethren . sit block give ... ''

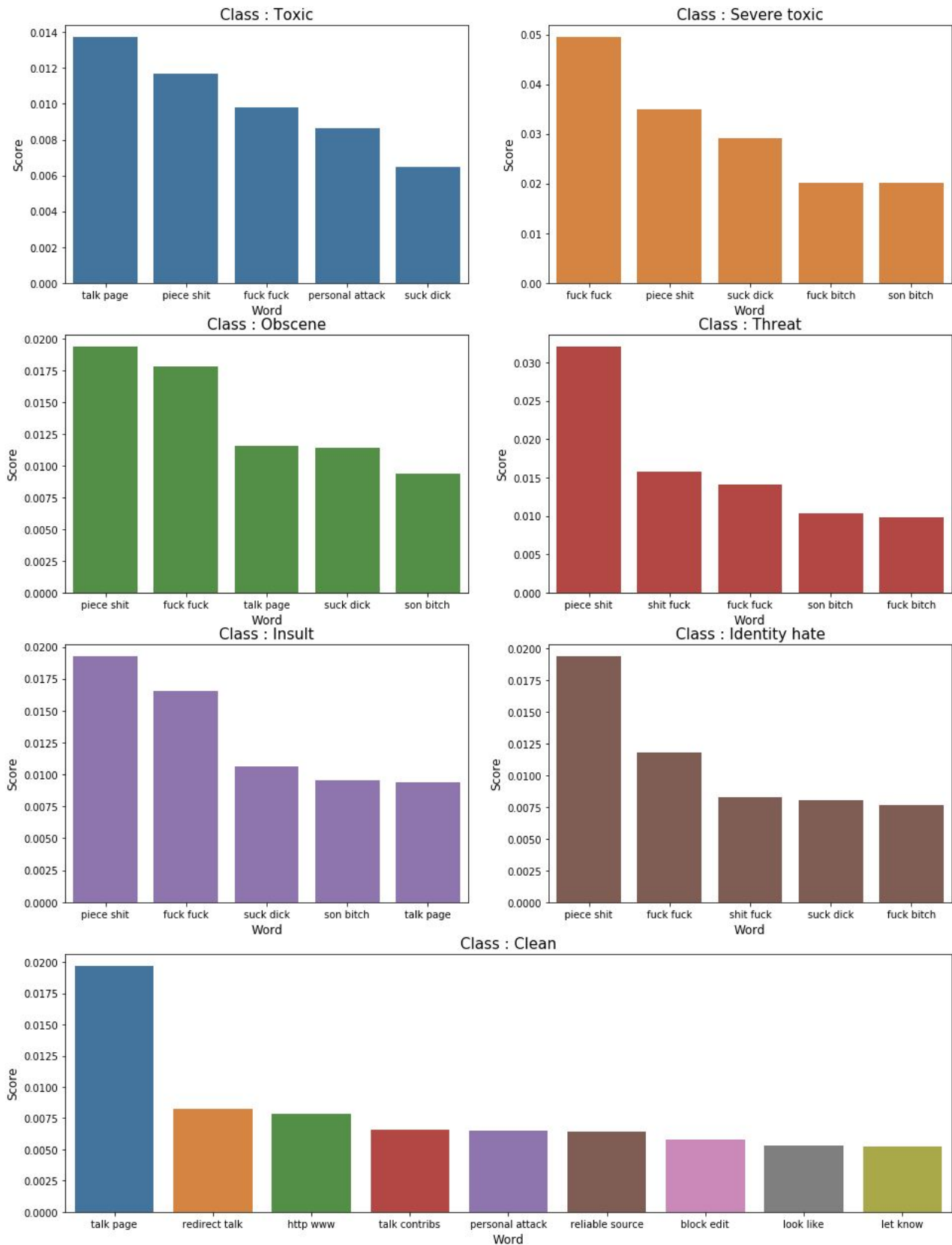
Total time of cleaning the data 193.79464602470398

After experimenting with with the cleaned data, and running it through the various counting methods outlined in the beginning of this section, the word frequency sets of unigrams and bigrams was created and the data displayed as follows:

Top words per class(unigrams)



Top words per class(Bigrams)

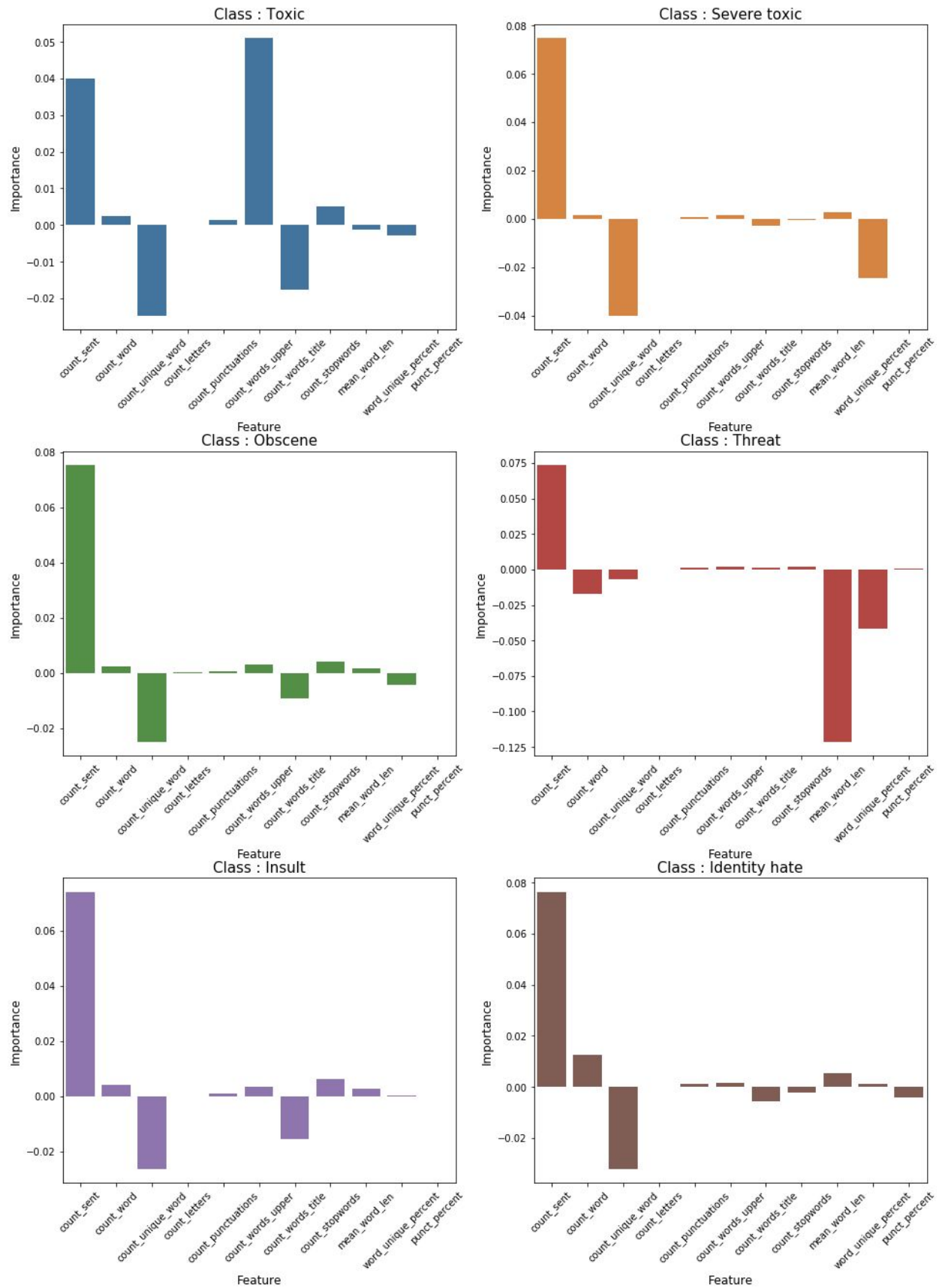


Upon inspecting the categorical data in the dataframe, the data loss for the the word frequencies was calculated to be as follows:

```
Using only Indirect features
Class:= toxic
Trainloss=log loss: 0.3009445063184722
Validloss=log loss: 0.3000824101795967
Class:= severe_toxic
Trainloss=log loss: 0.05048010934449732
Validloss=log loss: 0.05166794164872536
Class:= obscene
Trainloss=log loss: 0.1997549812663233
Validloss=log loss: 0.19806102599110711
Class:= threat
Trainloss=log loss: 0.01977152040060687
Validloss=log loss: 0.019228695141152517
Class:= insult
Trainloss=log loss: 0.18859304585564415
Validloss=log loss: 0.18981868052160533
Class:= identity_hate
Trainloss=log loss: 0.047831964459827736
Validloss=log loss: 0.05133031765313224
mean column-wise log loss:Train dataset 0.13456268794089526
mean column-wise log loss:Validation dataset 0.13503151185588655
Total time for Indirect feat model 536.8939189910889
```

After analyzing the strengths and weaknesses of each method, I determined what the optimal Feature Importance were by analyzing and displaying each features percent of loss in building the model to accurately predict the comment classification for each tag:

Feature importance for indirect features



Using the above data gives us all the features we need to implement the final Baseline Model for logistic regression to classify the data.

```
from scipy.sparse import csr_matrix, hstack

#Using all direct features
print("Using all features except the leaky ones")
target_x = hstack((train_bigrams,train_charngrams,train_unigrams,train_feats[SELECTED_COLS])).tocsr()

end_time=time.time()
print("total time till Sparse mat creation",end_time-start_time)
```

Using all features except the leaky ones
total time till Sparse mat creation 880.9988832473755

```
model = NbSvmClassifier(C=4, dual=True, n_jobs=-1)
X_train, X_valid, y_train, y_valid = train_test_split(target_x, target_y, test_size=0.33, random_state=2018)
train_loss = []
valid_loss = []
preds_train = np.zeros((X_train.shape[0], len(y_train)), dtype=np.float64)
preds_valid = np.zeros((X_valid.shape[0], len(y_valid)), dtype=np.float64)
for i, j in enumerate(TARGET_COLS):
    print('Class:= ' +j)
    model.fit(X_train,y_train[j])
    preds_valid[:,i] = model.predict_proba(X_valid)[: ,1]
    preds_train[:,i] = model.predict_proba(X_train)[: ,1]
    train_loss_class=log_loss(y_train[j],preds_train[:,i])
    valid_loss_class=log_loss(y_valid[j],preds_valid[:,i])
    print('Trainloss=log loss:', train_loss_class)
    print('Validloss=log loss:', valid_loss_class)
    train_loss.append(train_loss_class)
    valid_loss.append(valid_loss_class)
print('Mean column-wise loss: Train dataset', np.mean(train_loss))
print('Mean column-wise loss: Validation dataset', np.mean(valid_loss))

end_time=time.time()
print("Total time for NB base model creation",end_time-start_time)
```

```
Class:= toxic
Trainloss=log loss: 0.12561558174055648
Validloss=log loss: 0.13257933724944718
Class:= severe_toxic
Trainloss=log loss: 0.032426374816528235
Validloss=log loss: 0.03664143646334287
Class:= obscene
Trainloss=log loss: 0.08052903631852576
Validloss=log loss: 0.08017508556190277
Class:= threat
Trainloss=log loss: 0.01027588864200188
Validloss=log loss: 0.014056367552859913
Class:= insult
Trainloss=log loss: 0.15245857322790307
Validloss=log loss: 0.17157725443389493
Class:= identity_hate
Trainloss=log loss: 0.0687957143719264
Validloss=log loss: 0.0763706812657443
Mean column-wise loss: Train dataset 0.07835019485290697
Mean column-wise loss: Validation dataset 0.08523336042119867
Total time for NB base model creation 1050.2857520580292
```


Interactive Queries

Interactive queries can be issued by typing a comment into the react applications add a comment form. The application will take whatever you type into the form to query the model using the TensorFlow.js library.



The screenshot shows a web interface for adding a comment. At the top, the text "Add a comment" is displayed in a large, bold, black font. Below this is a text input field with a light gray border and a light gray background. Inside the input field, the placeholder text "Type a comment like this one" is written in a light gray font. Below the input field are two buttons. The first button is dark blue with the text "Check if your comment is toxic" in white. The second button is light blue with the text "Clear Comment" in white.

The queried model will return a percentage of classification for each of the tag groups clean, toxic, severe toxic, identify hate, threat, and insult as well as the predicted level of toxicity as seen below.

Tacos	1%	✓	Comment is clean
Party time!	0%	✓	Comment is clean
Go fuck yourself	95%	🚫	Comment clasified as threat
go to hell	90%	🚫	Comment clasified as identity_attack

Machine Learning

The product implements Feature Engineering, and Feature Importance to create a baseline model then uses logistic regression to determine the probability that a comment is toxic or not and to which category it falls into.

Accuracy Evaluation

Accuracy is evaluated by calculating the amount of loss between the training and testing datasets. Given that the lower the loss, the better the model and that the average training loss between datasets is less than 9%, the model appears to be quite accurate.

```
Using all features except the leaky ones
total time till Sparse mat creation 880.9988832473755

Class:= toxic
Trainloss=log loss: 0.12561558174055648
Validloss=log loss: 0.13257933724944718
Class:= severe_toxic
Trainloss=log loss: 0.032426374816528235
Validloss=log loss: 0.03664143646334287
Class:= obscene
Trainloss=log loss: 0.08052903631852576
Validloss=log loss: 0.08017508556190277
Class:= threat
Trainloss=log loss: 0.01027588864200188
Validloss=log loss: 0.014056367552859913
Class:= insult
Trainloss=log loss: 0.15245857322790307
Validloss=log loss: 0.17157725443389493
Class:= identity_hate
Trainloss=log loss: 0.0687957143719264
Validloss=log loss: 0.0763706812657443
Mean column-wise loss: Train dataset 0.07835019485290697
Mean column-wise loss: Validation dataset 0.08523336042119867
Total time for NB base model creation 1050.2857520580292
```

Security

A logging system is implemented to track whenever the program is in use by logging the time and server url. Because this product is meant to be used by multiple clients all connecting from different servers, the log file can be sorted by each customers dashboard to determine how often it is being used.

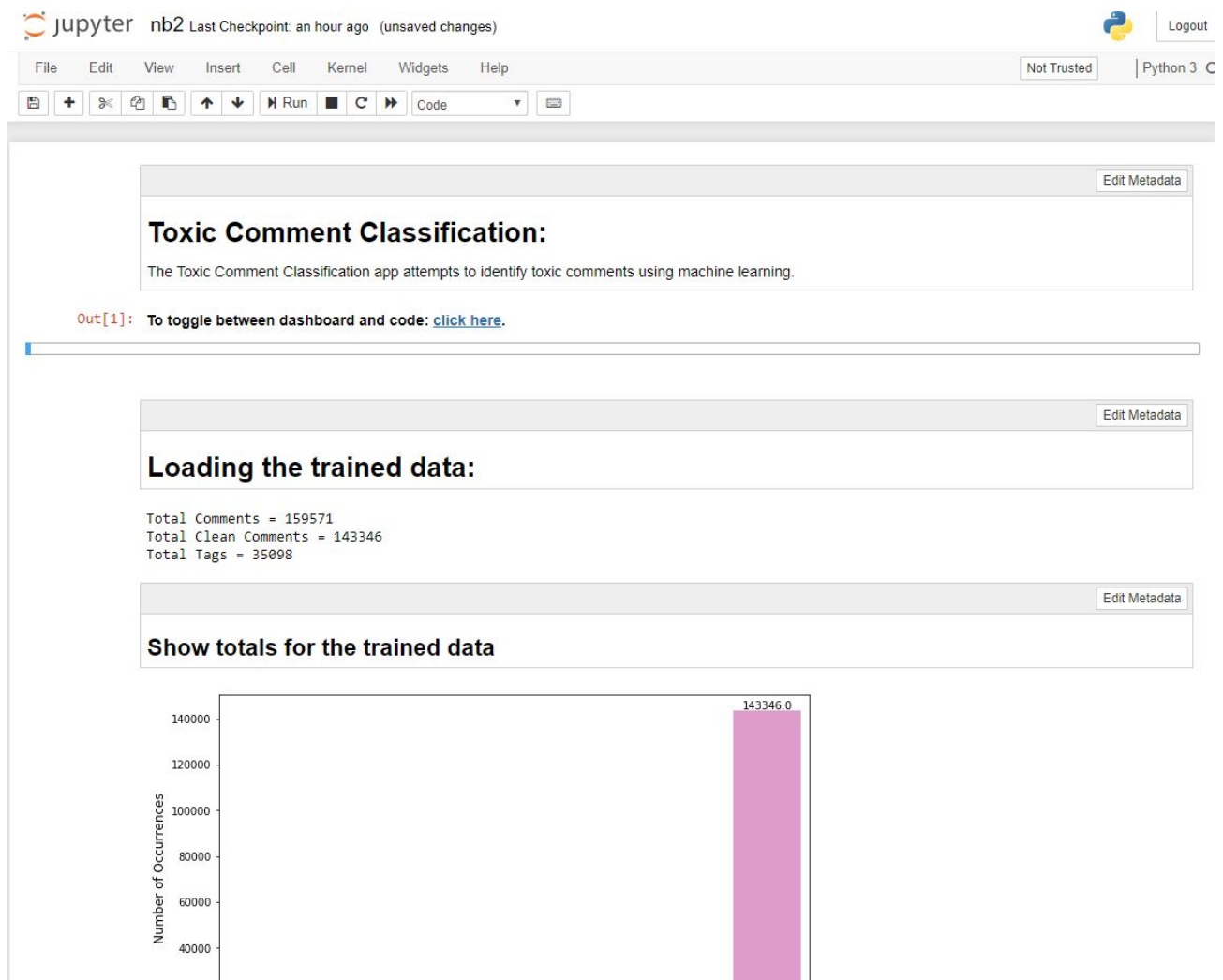
See the log.txt to view the log file.


Monitor and Maintenance


The model requires monthly update and user manual input on classification scoring to continue to improve the predictive. Once the data pipeline begins to add comments that were flagged as toxic, whether testing determined the comment was toxic or not will help implement reinforcement learning over time. The code was designed to be modular and allow frequent updates and monitoring the train and testing csv files.

Dashboard


The default view for the dashboard provides the main data visualization graphics and text with brief explanations of each section.




jupyter nb2 Last Checkpoint: an hour ago (autosaved)

 Logout

File Edit View Insert Cell Kernel Widgets Help
 Not Trusted | Python 3



Edit Metadata

Toxic Comment Classification:

The Toxic Comment Classification app attempts to identify toxic comments using machine learning.

In [1]:

Edit Metadata

```

from IPython.display import HTML, display

HTML('''<script>
code_show=true;
function code_toggle() {
  if (code_show){
    $('div.input').hide();
  } else {
    $('div.input').show();
  }
  code_show = !code_show
}
$( document ).ready(code_toggle);
</script><strong>To toggle between dashboard and code: <a href="javascript:code_toggle()">click here</a>.</strong>
</script>''')

```

Out[1]: To toggle between dashboard and code: [click here](#).

In [54]:

Edit Metadata

```

import pandas as pd
import numpy as np

import time
import gc
import warnings

from imageio import imread
from scipy import sparse
import scipy.stats as ss

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
from PIL import Image
import matplotlib_venn as venn

```

There is an interactive dashboard for the React application where you can interact with the data and see results in real time. The histogram will analyze the data of each previous submission to classify the number of toxic versus non toxic comments that have been posted.

Add a comment

Check if your comment is toxic
Clear Comment

Your comment	% of toxicity	🟢🚫	Label	Delete comment	Delete all comments
Taco	1%	🟢	Comment is clean	<button>Delete</button>	<button>Delete all</button>

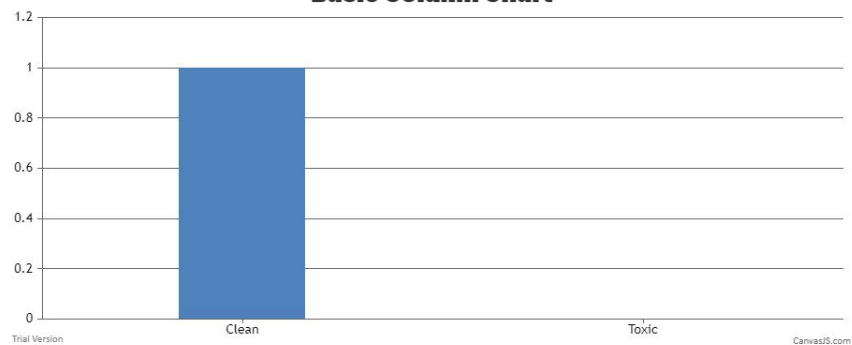
Basic Column Chart

The chart on the bottom of the screen only analyzes the data at the time of posting a comment. meaning that when you post a new comment, the current set of data is analyzed and the graph updated. So the new comment that you just posted will not show up in the graph until you post the next item as shown below.

Add a comment

Check if your comment is toxic
Clear Comment

Taco	1%	🟢	Comment is clean	<button>Delete</button>	<button>Delete all</button>
go die	80%	🚫	Comment clasified as insult	<button>Delete</button>	<button>Delete all</button>

Basic Column Chart

Section D – Documentation

Business Requirements

1. The product shall classify the predicted risk and category of a comment being toxic.
2. The product shall flag toxic comments upon posting according to the threshold set by the social media platform or online form.
3. The product helps online communities in preventing toxic environments and cyberbullying.

Raw and Cleaned Datasets

The raw datasets were obtained from Kaggle - Toxic Comment Classification Challenge (Jigsaw 2017) <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data>. The cleaned data sets were created in the code using corpus cleaning techniques.

Cleaning data:

Example corpus data:

```
In [42]: corpus.iloc[12235]
```

```
Out[42]: '''\n\n NOTE If you read above, and follow the links, any reader can see that I cited correctly the links I
added on this subject. Vidkun has added anotations to make them read as the oposite, but these links show
the ""official"" line taken by UGLE. I will not be trapped by any User into so-called 3RR, so he can peddl
e his POV. Strangly, ALL other ""MASONS"" are quiet, leaving "me" to defend that factual truth on my ow
n. ""Thanks"" Brethren. Sitting any blocking out if given... '''
```

Cleaned corpus data:

```
In [43]: clean(corpus.iloc[12235])
```

```
Out[43]: ''' note read , follow link , reader see cite correctly link add subject . vidkun add anotations make read
oposite , link show " " official " " line take ugle . trap user so-called 3rr , peddle pov . strangly , "
" masons " " quiet , leave ' ' ' defend factual truth . " " thank " " brethren . sit block give ... '''
```

```
In [44]: clean_corpus=corpus.apply(lambda x :clean(x))

end_time=time.time()
print("Total time of cleaning the data",end_time-start_time)
```

```
Total time of cleaning the data 193.79464602470398
```

Leaky variables were identified in the data sets and removed from the final data model.

Cleaning Leaky Data

Caution: Including this data can lead to unexpected results, it will not make sense to add them into the final model, so the data will be cleaned.

Here we are creating our own custom count vectorizer to create count variables that match our regex condition

```
In [30]: #Leaky Data
df['ip']=df["comment_text"].apply(lambda x: re.findall("\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}",str(x)))
#count of ip addresses
df['count_ip']=df["ip"].apply(lambda x: len(x))

#Find all the links
df['link']=df["comment_text"].apply(lambda x: re.findall("http://.*com",str(x)))
#get the number of links
df['count_links']=df["link"].apply(lambda x: len(x))

#article ids
df['article_id']=df["comment_text"].apply(lambda x: re.findall("\d:\d\d\d{s{0,5}}$",str(x)))
df['article_id_flag']=df['article_id'].apply(lambda x: len(x))

# Removing the usernames with a regex
df['username']=df["comment_text"].apply(lambda x: re.findall("\[User(.*)\]",str(x)))
# Count the number of username mentions
df['count_usernames']=df["username"].apply(lambda x: len(x))

# Leaky Ips
cv = CountVectorizer()
count_feats_ip = cv.fit_transform(df["ip"].apply(lambda x : str(x)))

# Leaky usernames
cv = CountVectorizer()
count_feats_user = cv.fit_transform(df["username"].apply(lambda x : str(x)))
```

Leaky Usernames

```
In [31]: # check names
cv.get_feature_names()[120:130]
```

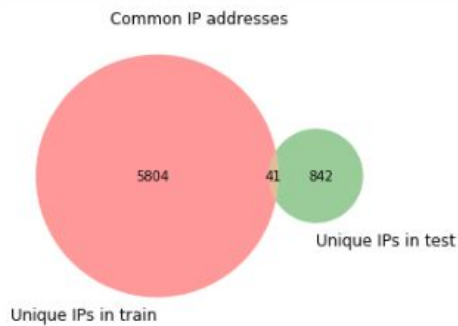
```
Out[31]: ['destruction',
'diablo',
'diligent',
'dland',
'dlohcierekim',
'dodo',
'dominick',
'douglas',
'dpl',
'dr']
```

```
In [32]: #Leaky Stability:
#Checking the re-occurrence of Leaky features to check their utility in predicting the test set.
leaky_feats=df[["ip","link","article_id","username","count_ip","count_links","count_usernames","article_id_flag"]]
leaky_feats_train=leaky_feats.iloc[:train.shape[0]]
leaky_feats_test=leaky_feats.iloc[train.shape[0]:]
```


Leaky IPs Count

```
In [33]: # filter out the data without ips
train_ips=leaky_feats_train.ip[leaky_feats_train.count_ip!=0]
test_ips=leaky_feats_test.ip[leaky_feats_test.count_ip!=0]
# get the unique ip list in test and train datasets
train_ip_list=list(set([a for b in train_ips.tolist() for a in b]))
test_ip_list=list(set([a for b in test_ips.tolist() for a in b]))

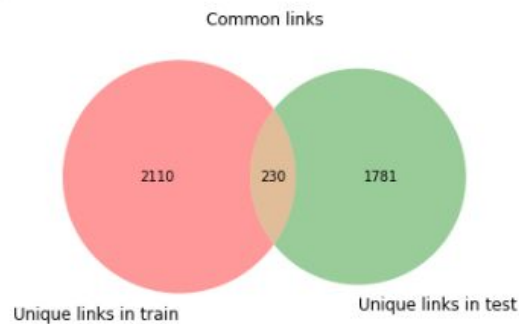
# get common elements
common_ip_list=list(set(train_ip_list).intersection(test_ip_list))
plt.title("Common IP addresses")
venn.venn2(subsets=(len(train_ip_list),len(test_ip_list),len(common_ip_list)),set_labels=("Unique IPs in t
rain","Unique IPs in test"))
plt.show()
```



Leaky Links Count

```
In [34]: #filter out the data without links
train_links=leaky_feats_train.link[leaky_feats_train.count_links!=0]
test_links=leaky_feats_test.link[leaky_feats_test.count_links!=0]
#get the unique list of ips in test and train datasets
train_links_list=list(set([a for b in train_links.tolist() for a in b]))
test_links_list=list(set([a for b in test_links.tolist() for a in b]))

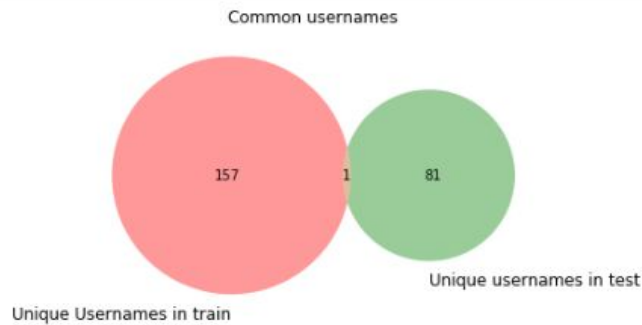
# get common elements
common_links_list=list(set(train_links_list).intersection(test_links_list))
plt.title("Common links")
venn.venn2(subsets=(len(train_links_list),len(test_links_list),len(common_links_list)), set_labels=("Uniqu
e links in train","Unique links in test"))
plt.show()
```



Leaky Usernames Count

```
In [35]: #filterout the entries without users
train_users=leaky_feats_train.username[leaky_feats_train.count_usernames!=0]
test_users=leaky_feats_test.username[leaky_feats_test.count_usernames!=0]
#get the unique list of ips in test and train datasets
train_users_list=list(set([a for b in train_users.tolist() for a in b]))
test_users_list=list(set([a for b in test_users.tolist() for a in b]))

# get common elements
common_users_list=list(set(train_users_list).intersection(test_users_list))
plt.title("Common usernames")
venn.venn2(subsets=(len(train_users_list),len(test_users_list),len(common_users_list)),
            set_labels=("Unique Usernames in train","Unique usernames in test"))
plt.show()
```



Finding blocked IPs

```
In [37]: train_ip_list=list(set([a for b in train_ips.tolist() for a in b]))
test_ip_list=list(set([a for b in test_ips.tolist() for a in b]))

# get common elements
blocked_ip_list_train=list(set(train_ip_list).intersection(blocked_ips))
blocked_ip_list_test=list(set(test_ip_list).intersection(blocked_ips))

print("There are",len(blocked_ip_list_train),"blocked IPs in train dataset")
print("There are",len(blocked_ip_list_test),"blocked IPs in test dataset")
```

```
There are 6 blocked IPs in train dataset
There are 0 blocked IPs in test dataset
```

```
In [38]: end_time=time.time()
print("total time cleaning Leaky Data",end_time-start_time)

total time cleaning Leaky Data 64.46692514419556
```

Code for Predictive Model

Predictive models begins after the raw data has been cleaned, indirect and direct features have been defined, and leaky features have been identified. The baseline model is then defined as seen below.

```
In [53]: # Baseline Model:
class NbsvmClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, C=1.0, dual=False, n_jobs=1):
        self.C = C
        self.dual = dual
        self.n_jobs = n_jobs

    def predict(self, x):
        # Verify that model has been fit
        check_is_fitted(self, ['_r', '_clf'])
        return self._clf.predict(x.multiply(self._r))

    def predict_proba(self, x):
        # Verify that model has been fit
        check_is_fitted(self, ['_r', '_clf'])
        return self._clf.predict_proba(x.multiply(self._r))

    def fit(self, x, y):
        # Check that X and y have correct shape
        y = y.values
        x, y = check_X_y(x, y, accept_sparse=True)

        def pr(x, y_i, y):
            p = x[y==y_i].sum(0)
            return (p+1) / ((y==y_i).sum()+1)

        self._r = sparse.csr_matrix(np.log(pr(x,1,y) / pr(x,0,y)))
        x_nb = x.multiply(self._r)
        self._clf = LogisticRegression(C=self.C, dual=self.dual, n_jobs=self.n_jobs).fit(x_nb, y)
        return self
```

After this, the data that will be used in the baseline models predictions is defined and evaluated against the indirect features.

```

In [54]: SELECTED_COLS=['count_sent', 'count_word', 'count_unique_word',
                        'count_letters', 'count_punctuations', 'count_words_upper',
                        'count_words_title', 'count_stopwords', 'mean_word_len',
                        'word_unique_percent', 'punct_percent']
target_x=train_feats[SELECTED_COLS]
# target_x

TARGET_COLS=['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
target_y=train_tags[TARGET_COLS]

In [55]: print("Using only Indirect features")
model = LogisticRegression(C=3)
X_train, X_valid, y_train, y_valid = train_test_split(target_x, target_y, test_size=0.33, random_state=2018)
train_loss = []
valid_loss = []
importance=[]
preds_train = np.zeros((X_train.shape[0], len(y_train)), dtype=np.float64)
preds_valid = np.zeros((X_valid.shape[0], len(y_valid)), dtype=np.float64)
for i, j in enumerate(TARGET_COLS):
    print('Class:= ' +j)
    model.fit(X_train,y_train[j])
    preds_valid[:,i] = model.predict_proba(X_valid)[: ,1]
    preds_train[:,i] = model.predict_proba(X_train)[: ,1]
    train_loss_class=log_loss(y_train[j],preds_train[:,i])
    valid_loss_class=log_loss(y_valid[j],preds_valid[:,i])
    print('Trainloss=log loss:', train_loss_class)
    print('Validloss=log loss:', valid_loss_class)
    importance.append(model.coef_)
    train_loss.append(train_loss_class)
    valid_loss.append(valid_loss_class)
print('mean column-wise log loss:Train dataset', np.mean(train_loss))
print('mean column-wise log loss:Validation dataset', np.mean(valid_loss))

end_time=time.time()
print("Total time for Indirect feat model",end_time-start_time)

```

From the indirect features, each feature's importance is calculated.

```

In [56]: importance[0][0]

Out[56]: array([ 3.98823934e-02,  2.47124288e-03, -2.46646880e-02,  1.99506683e-04,
                 1.36123468e-03,  5.09805921e-02, -1.77109252e-02,  5.01098837e-03,
                 -1.26865148e-03, -2.85927458e-03, -3.39294272e-05])

```

Using the calculates the indirect features are calculated and plotted.

```

In [57]: plt.figure(figsize=(16,22))
plt.suptitle("Feature importance for indirect features",fontsize=20)
gridspec.GridSpec(3,2)
plt.subplots_adjust(hspace=0.4)
plt.subplot2grid((3,2),(0,0))
sns.barplot(SELECTED_COLS,importance[0][0],color=color[0])
plt.title("Class : Toxic",fontsize=15)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Importance', fontsize=12)

plt.subplot2grid((3,2),(0,1))
sns.barplot(SELECTED_COLS,importance[1][0],color=color[1])
plt.title("Class : Severe toxic",fontsize=15)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Importance', fontsize=12)

plt.subplot2grid((3,2),(1,0))
sns.barplot(SELECTED_COLS,importance[2][0],color=color[2])
plt.title("Class : Obscene",fontsize=15)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Importance', fontsize=12)

plt.subplot2grid((3,2),(1,1))
sns.barplot(SELECTED_COLS,importance[3][0],color=color[3])
plt.title("Class : Threat",fontsize=15)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Importance', fontsize=12)

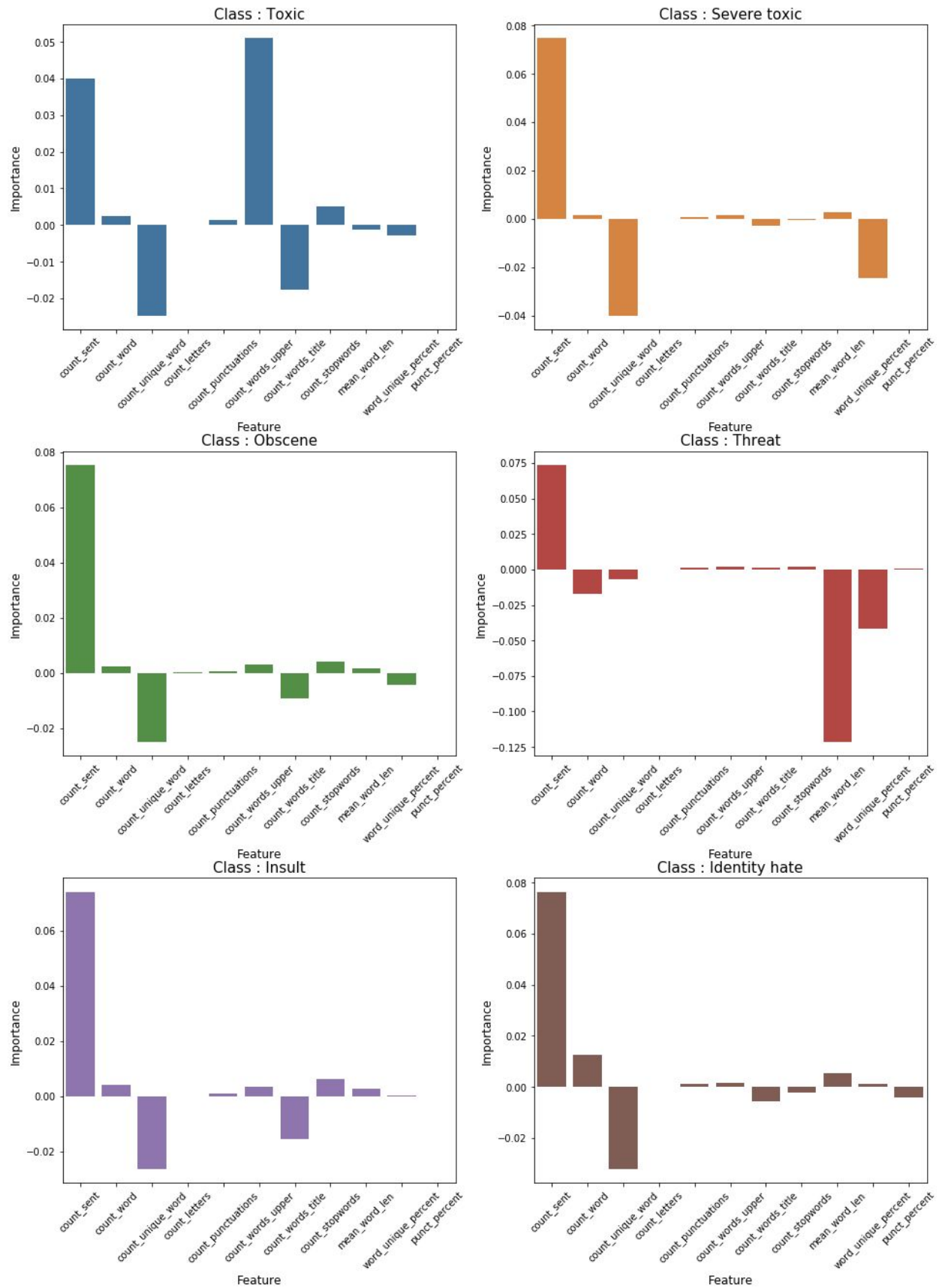
plt.subplot2grid((3,2),(2,0))
sns.barplot(SELECTED_COLS,importance[4][0],color=color[4])
plt.title("Class : Insult",fontsize=15)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Importance', fontsize=12)

plt.subplot2grid((3,2),(2,1))
sns.barplot(SELECTED_COLS,importance[5][0],color=color[5])
plt.title("Class : Identity hate",fontsize=15)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)
plt.xlabel('Feature', fontsize=12)
plt.ylabel('Importance', fontsize=12)

plt.show()

```


Feature importance for indirect features



A sparse matrix is created using the direct features.

```
In [58]: from scipy.sparse import csr_matrix, hstack

#Using all direct features
print("Using all features except the leaky ones")
target_x = hstack((train_bigrams,train_charngrams,train_unigrams,train_feats[SELECTED_COLS])).tocsr()

end_time=time.time()
print("total time till Sparse mat creation",end_time-start_time)

Using all features except the leaky ones
total time till Sparse mat creation 542.3741960525513
```

Using the data gathered, linear regression is then performed on the baseline model to create the final Naive Bayes model used for classification of the data.

```
In [59]: model = NbSvmClassifier(C=4, dual=True, n_jobs=-1)
X_train, X_valid, y_train, y_valid = train_test_split(target_x, target_y, test_size=0.33, random_state=2018)
train_loss = []
valid_loss = []
preds_train = np.zeros((X_train.shape[0], len(y_train)), dtype=np.float64)
preds_valid = np.zeros((X_valid.shape[0], len(y_valid)), dtype=np.float64)
for i, j in enumerate(TARGET_COLS):
    print('Class:= '+j)
    model.fit(X_train,y_train[j])
    preds_valid[:,i] = model.predict_proba(X_valid)[:,:1]
    preds_train[:,i] = model.predict_proba(X_train)[:,:1]
    train_loss_class=log_loss(y_train[j],preds_train[:,i])
    valid_loss_class=log_loss(y_valid[j],preds_valid[:,i])
    print('Trainloss=log loss:', train_loss_class)
    print('Validloss=log loss:', valid_loss_class)
    train_loss.append(train_loss_class)
    valid_loss.append(valid_loss_class)
print('Mean column-wise loss: Train dataset', np.mean(train_loss))
print('Mean column-wise loss: Validation dataset', np.mean(valid_loss))

end_time=time.time()
print("Total time for NB base model creation",end_time-start_time)
```

Hypothesis Assessment

Hypothesis: A predictive model can identify toxic comments with statistical significance.

The model trained on the Indirect features and achieved the following results.

```

Using only Indirect features
Class:= toxic
Trainloss=log loss: 0.3009445063184722
Validloss=log loss: 0.3000824101795967
Class:= severe_toxic
Trainloss=log loss: 0.05048010934449732
Validloss=log loss: 0.05166794164872536
Class:= obscene
Trainloss=log loss: 0.1997549812663233
Validloss=log loss: 0.19806102599110711
Class:= threat
Trainloss=log loss: 0.01977152040060687
Validloss=log loss: 0.019228695141152517
Class:= insult
Trainloss=log loss: 0.18859304585564415
Validloss=log loss: 0.18981868052160533
Class:= identity_hate
Trainloss=log loss: 0.047831964459827736
Validloss=log loss: 0.05133031765313224
mean column-wise log loss:Train dataset 0.13456268794089526
mean column-wise log loss:Validation dataset 0.13503151185588655
Total time for Indirect feat model 536.8939189910889

```

When the model was trained with the direct features, the following result was achieved.

```

Class:= toxic
Trainloss=log loss: 0.12878831252761466
Validloss=log loss: 0.13547211586837898
Class:= severe_toxic
Trainloss=log loss: 0.03504568971966303
Validloss=log loss: 0.04041251988779072
Class:= obscene
Trainloss=log loss: 0.07731113605197786
Validloss=log loss: 0.07783622445479192
Class:= threat
Trainloss=log loss: 0.013151504399634011
Validloss=log loss: 0.016624004271865122
Class:= insult
Trainloss=log loss: 0.1397365924673243
Validloss=log loss: 0.1603253192540611
Class:= identity_hate
Trainloss=log loss: 0.027856737565873153
Validloss=log loss: 0.038558003476059324
Mean column-wise loss: Train dataset 0.07031499545534785
Mean column-wise loss: Validation dataset 0.07820469786882452
Total time for NB base model creation 653.7177400588989

```

The hypothesis is thereby accepted as it shows promise that a model can generalize the prediction of toxic comments to data it was not trained with with a lower amount of training loss.

Visualization and Data Explorations

The data visualizations and explorations was covered in depth in the Data Preparation, Exploration and Visualization section of part B.

Accuracy Assessment

The product's accuracy is determined by the results achieved on the test dataset as follows:

```
Class:= toxic
Trainloss=log loss: 0.12878831252761466
Validloss=log loss: 0.13547211586837898
Class:= severe_toxic
Trainloss=log loss: 0.03504568971966303
Validloss=log loss: 0.04041251988779072
Class:= obscene
Trainloss=log loss: 0.07731113605197786
Validloss=log loss: 0.07783622445479192
Class:= threat
Trainloss=log loss: 0.013151504399634011
Validloss=log loss: 0.016624004271865122
Class:= insult
Trainloss=log loss: 0.1397365924673243
Validloss=log loss: 0.1603253192540611
Class:= identity_hate
Trainloss=log loss: 0.027856737565873153
Validloss=log loss: 0.038558003476059324
Mean column-wise loss: Train dataset 0.07031499545534785
Mean column-wise loss: Validation dataset 0.07820469786882452
Total time for NB base model creation 653.7177400588989
```

This shows a mean average of 7% data loss in the comment classification, specificity 93% accuracy for the predictive model.

Application Testing

Unit testing was performed on the React application to ensure that queries can be performed in real-time using different environments.

The first iteration of the baseline model had some calculation errors. I used a float32 to try to save space in calculation the data loss of the model and got the following result:

[old result]

```

Class:= toxic
Trainloss=log loss: nan
Validloss=log loss: nan
Class:= severe_toxic
Trainloss=log loss: inf
Validloss=log loss: inf
Class:= obscene
Trainloss=log loss: nan
Validloss=log loss: nan
Class:= threat
Trainloss=log loss: inf
Validloss=log loss: 0.013863400046199634
Class:= insult
Trainloss=log loss: nan
Validloss=log loss: nan
Class:= identity_hate
Trainloss=log loss: nan
Validloss=log loss: nan
Mean column-wise loss: Train dataset nan
Mean column-wise loss: Validation dataset nan
Total time for NB base model creation 815.0054860115051

```

After much trial and error and tweaking the code, I found that using a float64 gives the desired result as seen below.

[new result]

```

Class:= toxic
Trainloss=log loss: 0.12878831252761466
Validloss=log loss: 0.13547211586837898
Class:= severe_toxic
Trainloss=log loss: 0.03504568971966303
Validloss=log loss: 0.04041251988779072
Class:= obscene
Trainloss=log loss: 0.07731113605197786
Validloss=log loss: 0.07783622445479192
Class:= threat
Trainloss=log loss: 0.013151504399634011
Validloss=log loss: 0.016624004271865122
Class:= insult
Trainloss=log loss: 0.1397365924673243
Validloss=log loss: 0.1603253192540611
Class:= identity_hate
Trainloss=log loss: 0.027856737565873153
Validloss=log loss: 0.038558003476059324
Mean column-wise loss: Train dataset 0.07031499545534785
Mean column-wise loss: Validation dataset 0.07820469786882452
Total time for NB base model creation 653.7177400588989

```

There was a lot more debugging and testing that had to be done, however this was the biggest pain point that ate up most of my time. I also noticed that when I would run the notebook on a Windows computer, it would encounter an out of memory error, but when it was run on a Mac, it would run successfully.

Source Code and Executable Files

```

Capstone/
  /public
    /index.html          -- required to run the react app
    /manifest.json
    /robots.txt
  /src
    /forms
      /AddCommentForm.js -- Add comment component
    /graphs
      /canvasjs.min.js
      /canvasjs.react.js -- library files for the toxic graph
      /jquery.canvasjs.js
      /ToxicGraph.js      -- toxic graph component
    /tables
      /ToxicTable.js      -- toxic table component
    /App.css
    /App.js               -- Main execution of the react app
    /App.test.js
    /Footer.js            --unused but contains links to project data
    /index.css
    /index.js
    /serviceWorker.js     --starts the react server
    /static.json
  /Toxic                  --Main project folder
    /src
      /log.log            -- log file
      /nb2.ipynb          -- Main project app
      /test.csv           -- test data for training the model
      /test_labels.csv    -- set of test labels for training the model
      /train.csv          -- data to train the model
    /package.json         -- package data for npm install of react app
    /package-lock.json

```

Quick Start Guide

Installation has only been tested on Mac platforms with either Anaconda installed; be sure to select the Python 3.7 version with either of the following:

STEP 1: Download and install Node.js and Anaconda from the following URLs:

Node.js - <https://nodejs.org/en/>

Anaconda – <https://www.anaconda.com/distribution/#download-section>

STEP 2: Extract the capstone project form the submitted zip folder.

STEP 3: Open a terminal window and enter the following commands:

- `easy_install matplotlib-venn`
- `pip install -U spacy`
- `sudo python -m nltk.downloader -d /usr/local/share/nltk_data all`
- `conda install jupyter_dashboards -c conda-forge`

STEP 4: Navigate to the Capstone using the terminal and enter the following command:

- `sudo npm install`
- `jupyter trust notebook.ipynb`

This will download and install the node modules need for the React.js application.

STEP 5: To run the React.js application, type the following command into the terminal:

- `npm start`

To run the Jupyter Notebook, open the Anacondas Dashboard and launch Jupyter Notebooks. Notebooks is loaded navigate to the location of where you have the extracted capstone folder. Click on the notebook.ipynb file to launch the Notebook.

STEP 6: If you followed the above steps correctly, you should now have two browser windows open. One for the notebook and one for the react application.

Running the Jupyter Notebook application:

Click run in each cell or click run all in the Jupyter Notebook to perform a new run of the testing and trained data and all features.

NOTE: this may take several minutes depending on how powerful your computer is.

NOTE: if you are using a windows computer to run this application, several of the cells will run out of memory before the final calculations can be completed. Do not run this on a windows computer.

Running the React.js Application:

To run the React.js application, simply type into the add a comment form and submit. The data will then be classified using the Tensorflow.js library and its predicted result returned and populated into the table on the right.

The toxic graph will update after submit with a calculation of the previous data in a bar graph.

NOTE: This means that the graph is always one comment behind. So, if you have posted two comments into the table only one will show on the graph.

Section E: References

Kaggle - Toxic Comment Classification Challenge (Jigsaw 2017)

<https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data>

General Data Protection Regulation GDPR (GDPR 2019) <https://gdpr-info.eu/>

<https://www.stopbullying.gov/cyberbullying/kids-on-social-media-and-gaming/index.html>

How Big of a Problem is Cyber Bullying Among Teens? (huffpost.com 2017)

https://www.huffpost.com/entry/how-big-of-a-problem-is-cyber-bullying-among-teens_b_58c4a397e4b0a797c1d39da3