Abusive Chat Detection:

Automatically Identifying Inappropriate Messages

Samuel P. Sears

Bellevue University

Abstract

In this project, labeled tweets are examined to find associations between words in the tweets and hate or offensive language. The data sets are then processed and vectorized as a means to create machine learning models that predict whether or not a message contains hate speech or offensive language. The models are then used in addition to a list of profane words to create a pipeline for screening messages for inappropriate language. The pipeline is then tested on a labeled random sample from a customer service chat data set. Finally, a use case for this service is explained for the customer service chat domain.

Keywords: machine learning, binary classification, pycaret, NLP

Abusive Chat Detection:

Automatically Identifying Inappropriate Messages

In the customer service domain, not all customers are pleasant when seeking support. Further, there's no support channel completely free from abusive customers (WhosOn, n.d.). Inappropriate messages can come from prospective customers, current customers, or even from within the company. Anger and frustration from the situation can quickly escalate conversations from distress to abuse. On the other side of that conversation is often a human begin just trying to do their job. They work tirelessly each and every day to answer complex and unique requests (Martinovic & Power, n.d.). When faced with inappropriate or abusive messages, they can be left not only unsure of what to do, but also unsafe. In some situations, the first instinct in responding to an insult or attack may be to defend yourself or retaliate (WhosOn, n.d.). However, this can often just escalate the situation more.

Business Problem and Hypothesis

If a company has a customer service channel, there's no way to guarantee abusive customers will not be present. In the chat channel, however, there are potential solutions. One solution might be to attempt to train employees on how to deal with these customers and de-escalate the situation or send the customer to a manager. However, this does not protect employees from abuse. Someone is still receiving the full force of the hateful or offensive communication. This can lead to decreased morale. A superior solution is to detect and automatically handle these chats. If chats are fielded by a chat bot, the messages could be screened prior to escalating the customer to a live representative. If a customer is using abusive language with a chat bot, there's no reason for the company to be sending that customer to one of their employees. Messages could be also screened within chat systems prior to displaying the message to a representative. Full integration of a screening service within a company's current system is out of scope for this project. The first step in solving this problem is to be able to reliably detect abusive chats. This project aims to create a proof of concept that is able to screen messages for hate speech, offensive language, and profanity.

Data

There were a few different data sets used throughout this project for the different goals of the service. The first two goals were to be able to predict whether a message contains hate speech or is offensive. For this goal, two datasets from Kaggle were used (Mobius, 2021). One data set contained 12,970 tweets labeled for whether or not they contained hate speech. The other data set contained 14,100 tweets labeled for whether or not they contained offensive language. Although tweets can be a bit different then customer service messages, they do share a similarity in being short messages. Due to this, and the fact that a labeled data set was difficult to come by for customer service messages, the tweets were used for training models to detect hate speech and offensive language. Another goal of the service was to detect profanity. For this, a list of profane language was simply downloaded from bannedwordlist.com. However, in a real use case, a list of profane language would need to be defined by the business.

The last data set used for this project was to ensure that the models had high precision on actual customer service messages. A company looking to implement this service might want to block abusive customers from continuing through their system. However, blocking a customer who was not hateful, offensive, or using profanity would likely be unacceptable. Any prevention of abusive customers would be an improvement for a company with no system in place, but blocking even a few customers by mistake could override any benefit. Therefore, a free dataset from Bitext was obtained (Bitext, 2019). This dataset contained over 20,000 customer service messages. A random sample of 299 messages was gathered and labeled for whether or not the messages contained hate speech, were offensive, or had profanity present. This sample was then set aside for a final test of the service.

Exploratory Data Analysis

To start out the project, the hate speech and offensive language data was loaded and examined. Unlike other machine learning projects where there are multiple predictor variables defined that explain a response variable, this project contained text which explained a response variable. Therefore, typical tasks of looking at distributions of predictor variables were not a part of exploring the data. Instead, the words in the text were examined to get an idea of which words were associated more with hate speech or offensive language. In order to see words that were more frequent within hate tweets and offensive tweets, both data frames were filtered to include only hate or offensive speech. Frequencies of words were then gathered for all words that were found in these tweets. In order to visualize the result, a word cloud was created. Word clouds are a technique used for representing text data where the size of each word in the visualization represents its frequency (GeeksforGeeks, 2021). The resulting visualizations highlighted the words that were most common in hate tweets and offensive tweets. These can be found in Appendix A.

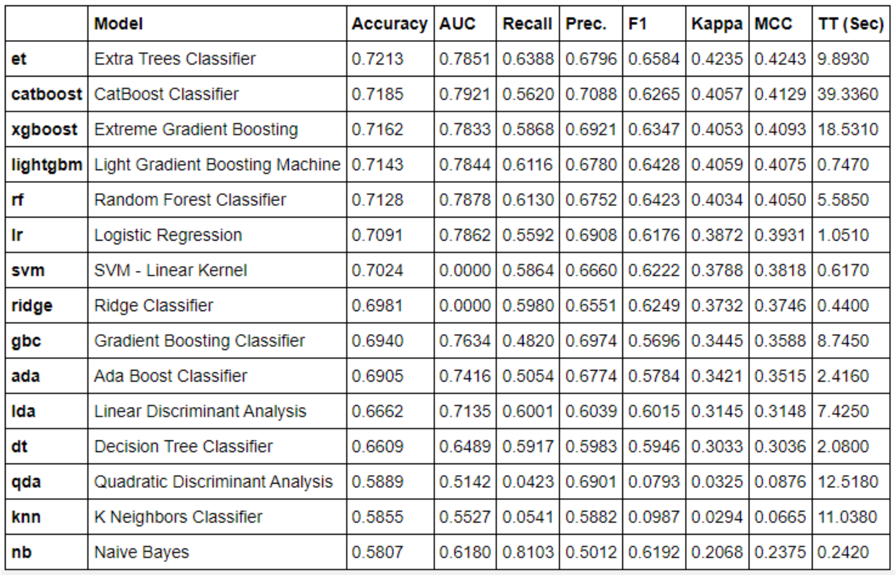
Predictive Modeling

Prior to fitting any model on the hate and offensive tweet data, there were several preprocessing steps that had to occur. The first preprocessing step was to remove any numbers or punctuation in the data. In the end, the model to be fit would be using words to determine whether the message contained hate speech or offensive speech. Removing the numbers and punctuation within the text served to normalize the words so there weren’t so many words for the model to consider. Converting all text to lowercase was also completed, but as a part of the vectorization technique in later steps. Next, the data was split between training and testing data. The testing data was withheld from any modeling steps to ensure there was unseen data to evaluate the models on. Lastly, the text data had to be converted into a format that the machine learning model could understand. One way to do this is with vectorization. When text data is vectorized, the predictor variables become either words or phrases and each row of the data contains a value indicating a score for that word or phrase. There are several vectorization techniques as well. One technique is to simply put the frequency of that word in the observation. This is referred to as a count vectorizer. A superior method is to use a term frequency – inverse document frequency (TFIDF) vectorizer which not only takes into account the frequency of the word in the observation, but also the entire text corpus (Saket, 2020). This was the vectorization technique chosen for this project. As a result, each message in the train and testing data sets were replaced by TFIDF vectors which represented the most frequent 2000 words in the training data set. For modeling, the data was now seen as a binary classification problem with 2000 numeric predictor variables.

To speed up the modeling process, and to ensure a wide variety of models were tested, the compare_models function from PyCaret's classification module was utilized (PyCaret, n.d.). An experiment was set up and the compare_models function was run to produce preliminary evaluation metrics for 15 different models on both the hate and offensive data as shown in Figure 1 and 2.

Figure 1

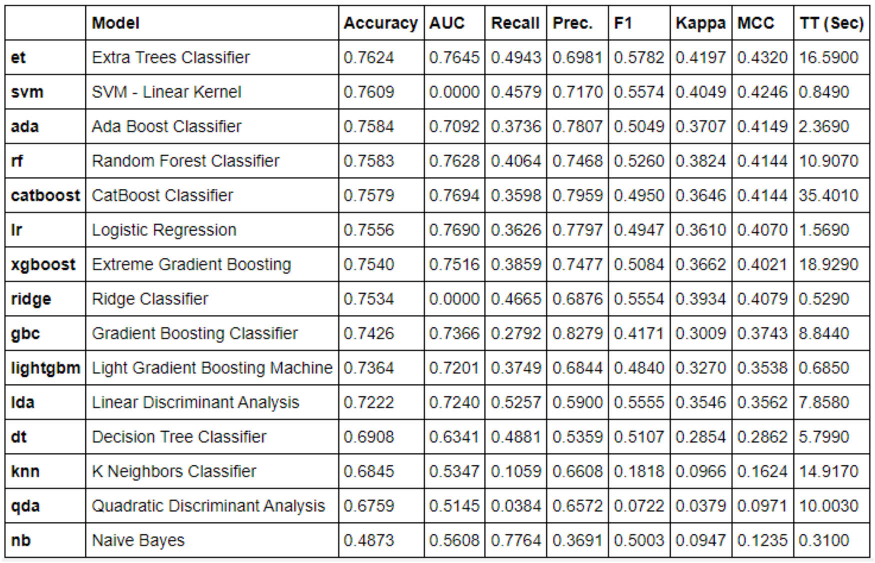
Results from compare_models function on hate data



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	0.7213	0.7851	0.6388	0.6796	0.6584	0.4235	0.4243	9.8930
catboost	CatBoost Classifier	0.7185	0.7921	0.5620	0.7088	0.6265	0.4057	0.4129	39.3360
xgboost	Extreme Gradient Boosting	0.7162	0.7833	0.5868	0.6921	0.6347	0.4053	0.4093	18.5310
lightgbm	Light Gradient Boosting Machine	0.7143	0.7844	0.6116	0.6780	0.6428	0.4059	0.4075	0.7470
rf	Random Forest Classifier	0.7128	0.7878	0.6130	0.6752	0.6423	0.4034	0.4050	5.5850
lr	Logistic Regression	0.7091	0.7862	0.5592	0.6908	0.6176	0.3872	0.3931	1.0510
svm	SVM - Linear Kernel	0.7024	0.0000	0.5864	0.6660	0.6222	0.3788	0.3818	0.6170
ridge	Ridge Classifier	0.6981	0.0000	0.5980	0.6551	0.6249	0.3732	0.3746	0.4400
gbc	Gradient Boosting Classifier	0.6940	0.7634	0.4820	0.6974	0.5696	0.3445	0.3588	8.7450
ada	Ada Boost Classifier	0.6905	0.7416	0.5054	0.6774	0.5784	0.3421	0.3515	2.4160
lda	Linear Discriminant Analysis	0.6662	0.7135	0.6001	0.6039	0.6015	0.3145	0.3148	7.4250
dt	Decision Tree Classifier	0.6609	0.6489	0.5917	0.5983	0.5946	0.3033	0.3036	2.0800
qda	Quadratic Discriminant Analysis	0.5889	0.5142	0.0423	0.6901	0.0793	0.0325	0.0876	12.5180
knn	K Neighbors Classifier	0.5855	0.5527	0.0541	0.5882	0.0987	0.0294	0.0665	11.0380
nb	Naive Bayes	0.5807	0.6180	0.8103	0.5012	0.6192	0.2068	0.2375	0.2420

Figure 2

Results from compare models function on offensive data



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
et	Extra Trees Classifier	0.7624	0.7645	0.4943	0.6981	0.5782	0.4197	0.4320	16.5900
svm	SVM - Linear Kernel	0.7609	0.0000	0.4579	0.7170	0.5574	0.4049	0.4246	0.8490
ada	Ada Boost Classifier	0.7584	0.7092	0.3736	0.7807	0.5049	0.3707	0.4149	2.3690
rf	Random Forest Classifier	0.7583	0.7628	0.4064	0.7468	0.5260	0.3824	0.4144	10.9070
catboost	CatBoost Classifier	0.7579	0.7694	0.3598	0.7959	0.4950	0.3646	0.4144	35.4010
lr	Logistic Regression	0.7556	0.7690	0.3626	0.7797	0.4947	0.3610	0.4070	1.5690
xgboost	Extreme Gradient Boosting	0.7540	0.7516	0.3859	0.7477	0.5084	0.3662	0.4021	18.9290
ridge	Ridge Classifier	0.7534	0.0000	0.4665	0.6876	0.5554	0.3934	0.4079	0.5290
gbc	Gradient Boosting Classifier	0.7426	0.7366	0.2792	0.8279	0.4171	0.3009	0.3743	8.8440
lightgbm	Light Gradient Boosting Machine	0.7364	0.7201	0.3749	0.6844	0.4840	0.3270	0.3538	0.6850
lda	Linear Discriminant Analysis	0.7222	0.7240	0.5257	0.5900	0.5555	0.3546	0.3562	7.8580
dt	Decision Tree Classifier	0.6908	0.6341	0.4881	0.5359	0.5107	0.2854	0.2862	5.7990
knn	K Neighbors Classifier	0.6845	0.5347	0.1059	0.6608	0.1818	0.0966	0.1624	14.9170
qda	Quadratic Discriminant Analysis	0.6759	0.5145	0.0384	0.6572	0.0722	0.0379	0.0971	10.0030
nb	Naive Bayes	0.4873	0.5608	0.7764	0.3691	0.5003	0.0947	0.1235	0.3100

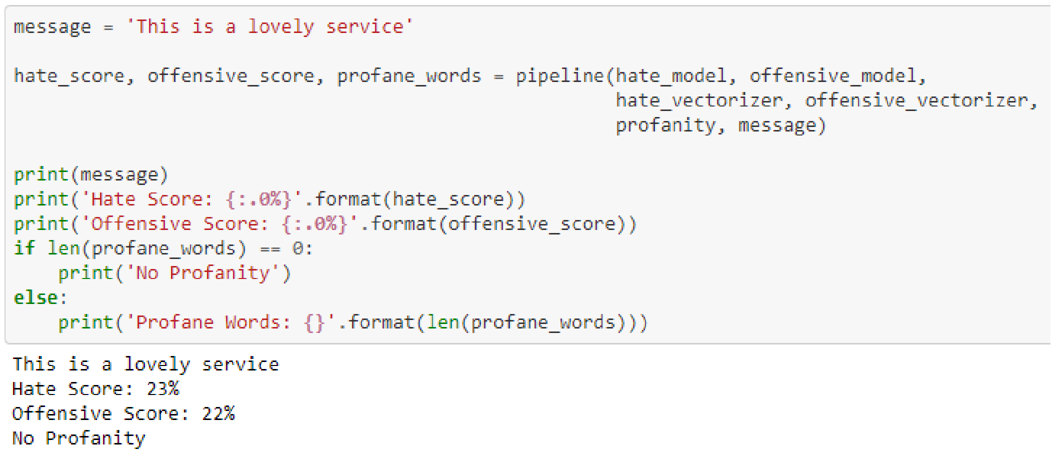
After reviewing these results, the CatBoost Classifier was chosen as the model for both the hate data and offensive data. It had the highest AUC score, which meant it was the best performing model in terms of separating the positive and negative classes at different model probability thresholds. The accuracy metrics only consider the probability threshold for predicting the positive class of 0.5. Since the probability threshold is something that can be adjusted in a business use case, the accuracy scores provided were not a comprehensive evaluation metric.

Application

Finally, it was time to put everything together into a pipeline that could be used in a business case. The goal of the project was to create a service that would take a message and return whether that message contained hate speech, offensive language, or profanity. Therefore, a pipeline was created that would mimic this system. The pipeline created was a function that contained four steps; preprocess the message, check for hate speech, check for offensive language, and check for profanity. The return of the pipeline was a score for likelihood the message contained hate speech, a score for the likelihood the message contained offensive language, and a list of the profane words found within the message. The preprocess function used the same preprocessing steps that were used prior to modeling. The check for hate speech and offensive language functions used the models and vectorizers created previously to create predictions scores for the message. Lastly, the profanity check screened the words within the message for any words in the profanity list passed to the pipeline. An example output for a message can be seen in Figure 3. In this example, a non-hateful, non-offensive message without profanity was sent through the pipeline and the returned variables are displayed.

Figure 3

Example output and code for abusive chat detection service pipeline



```
message = 'This is a lovely service'

hate_score, offensive_score, profane_words = pipeline(hate_model, offensive_model,
                                                      hate_vectorizer, offensive_vectorizer,
                                                      profanity, message)

print(message)
print('Hate Score: {:.0%}'.format(hate_score))
print('Offensive Score: {:.0%}'.format(offensive_score))
if len(profane_words) == 0:
    print('No Profanity')
else:
    print('Profane Words: {}'.format(len(profane_words)))
```

```
This is a lovely service
Hate Score: 23%
Offensive Score: 22%
No Profanity
```

In a business use case, a threshold could be defined for hate score and offensive score in which messages that obtained scores above a certain threshold could be handled differently. For example, they conversation could be concluded with a message to the user that hate speech and offensive language is prohibited. However, that may invoke a sense of unease with a business. They may be reluctant to let a machine learning algorithm make the decision to block a customer from talking with their support reps if there is a chance the algorithm would incorrectly classify a message as hate speech or offensive when it was not. Therefore, a test of precision was needed.

In order to ensure the models were not falsely predicting high hate and offensive scores on non-offensive chat messages, a test was conducted on the labeled samples from the BiText dataset mentioned previously. All 299 chat messages were sent through the pipeline and the returned variables were tracked. Hate or offensive scores of 0.5 or greater were set as positive predictions that the message contained hate or offensive speech. If the list of the profane words found in the message was greater than 0, it was recorded that the message contained profanity. When comparing the actual labels with the predicted labels, hate predictions, offensive predictions, and profanity detection had 100% accuracy. This was especially promising because only 4% of the labeled data set contained offensive messages and profanity. No messages contained hate speech. So, not only was the model accurate, the model did not incorrectly flag messages for hate speech, offensive speech, or profanity even when the overwhelming majority of the data was not any of these. In other words, the models were given many chances to make false positive predictions, but passed the test every time.

Further Analysis

This project was done as a proof of concept that abusive chats can be automatically detected in the customer service space using machine learning. Even by training models on tweet data, high levels of accuracy were seen. The next step for a company wanting to utilize a service like this would start with training models on labeled chat data from within their organization. At the very least, the company would need to test the models on their own chat data to ensure the same high levels of accuracy could be obtained. Next, the company would have to check to see where this service could be integrated within their current system. An example of how this could be integrated might be to call the service through an API. The company would also need to tune the model based on the thresholds they were comfortable with. If they set the thresholds lower, there would be a higher likelihood of a chat being classified as offensive or hateful. If they set the thresholds higher, there'd be a lesser chance. Lastly, the company would need to review and provide a list of profanity they deemed unacceptable.

References

BiText. (2019, Dec 26). *Bitext's Customer Support Dataset for free.* *https://blog.bitext.com/free-customer-support-dataset*

GeeksforGeeks. (2021, July 5). *Generating Word Cloud in Python. https://www.geeksforgeeks.org/generating-word-cloud-python/*

Martinovic, F. & Power, C. (n.d.) *Custting the cord on inappropriate customer conversations.* Intercom. https://www.intercom.com/blog/how-to-cut-the-cord-on-inappropriate-customer-conversations/

Mobius. (2021). *Seven NLP Tasks With Twitter Datasets.* Kaggle. https://www.kaggle.com/arashnic/7-nlp-tasks-with-tweets

PyCaret. (n.d.). *Compare Models*. https://pycaret.org/compare-models/

Saket, S. (2020, Jan 12). *Count Vectorizer vs TFIDF Vectorizer.* LinkedIn. https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket

WhosOn. (n.d). *When chatters attack: dealing with abusive customers*. https://www.whoson.com/customer-service/when-chatters-attack-dealing-with-abusive-customers/

Appendix A

Word Clouds

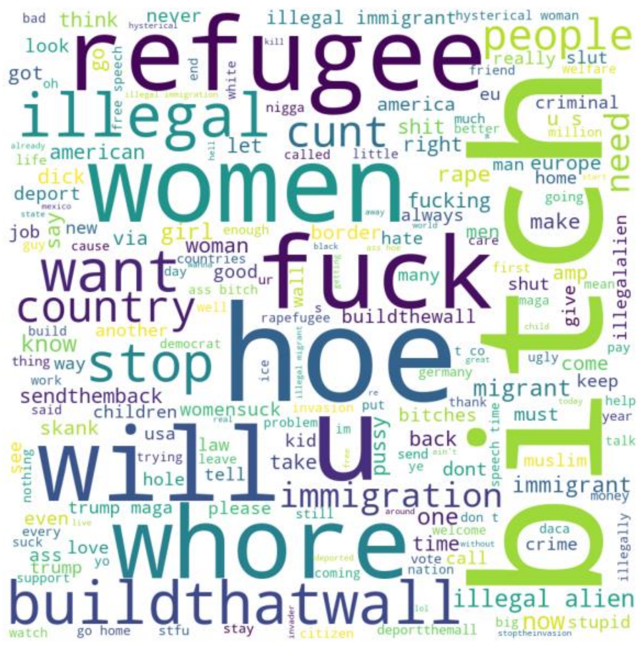


Figure A1. Word cloud of tweets that contained hate speech.

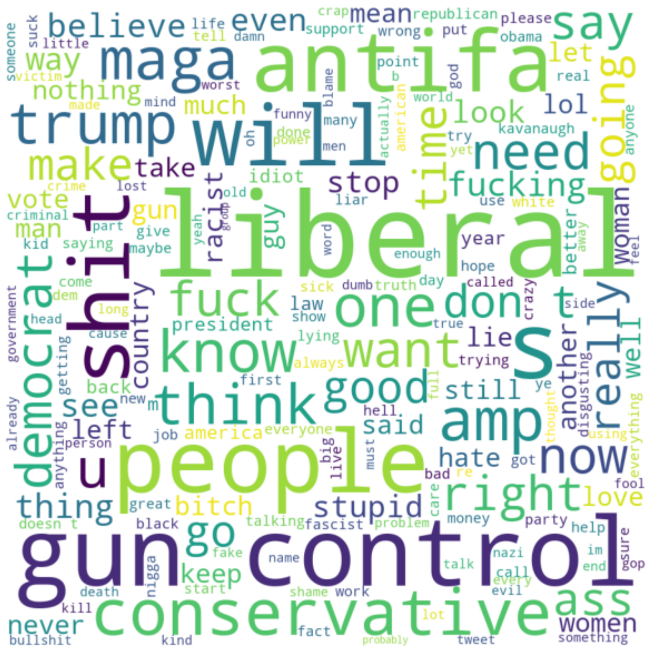


Figure A2. Word cloud of tweets that contained offensive language.