**Toxic Comment Classification**

Aditya Kanungo  
26th April 2018

**Contents**

**1 Introduction**  
 1.1 Problem Statement   
 1.2 Data

**2 Methodology**

2.1 Pre-Processing

2.1.1 Missing value-analysis  
 2.1.2 General text comments study

2.1.3 Visualization of dataset  
 2.1.4 Data Cleaning  
 2.1.5 Word Cloud to category wise visualize frequent comments  
  
 2.2 Vectorization of data  
 2.2.1 Result of vectorizing train dataset  
 2.2.2 Result of vectorizing test dataset  
  
 2.3 Modeling  
 2.3.1 Model Selection  
 2.3.2 Multi-label classification

**3 Conclusion**

3.1 Model Evaluation

**Appendix – Python Code  
References**

**Chapter 1**

**Introduction**

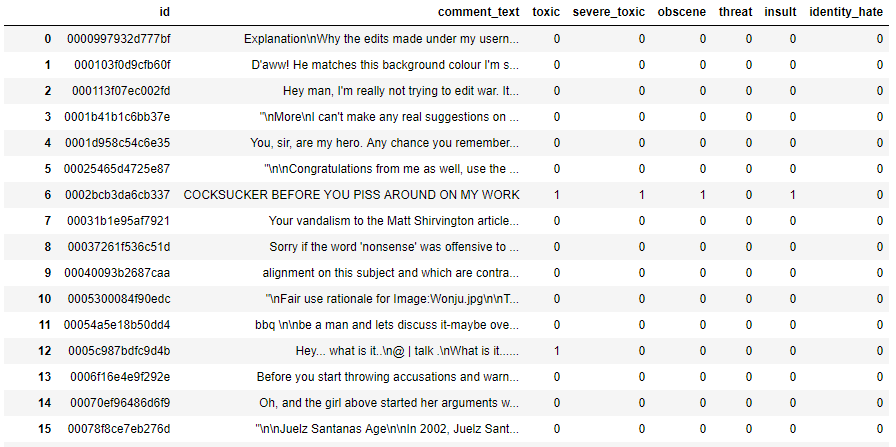
* 1. **Problem Statement**

Discussing things, you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. So, we have to build a multi-headed model that’s capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate. You’ll be using a dataset of comments from Wikipedia’s talk page edits. it would hopefully help online discussion become more productive and respectful.

* 1. **Data**

Our task is to build a model which will predict the probability of a comment to be under the different types of toxicity like threats, obscenity, insults, and identity-based hate.

* + 1. **Train Dataset:** (id, comment\_text, toxic, severe\_toxic, obscene, threat, insult, identity\_hate)  
       (159571 rows × 8 columns)



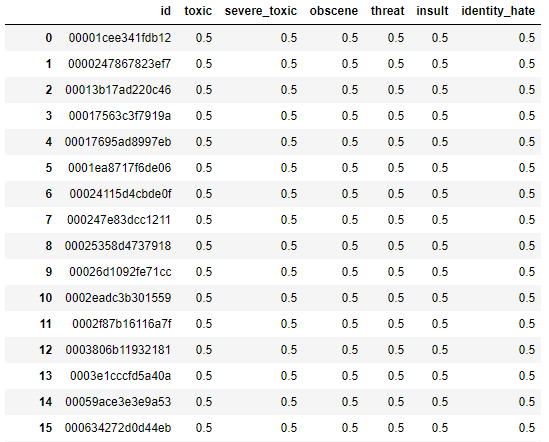
* + 1. **Test Dataset:** (id, comment\_text)

(153164 rows x 2 columns)



* + 1. **Sample submission:** (id, toxic, severe\_toxic, obscene, threat, insult, identity\_hate)

(153164 rows × 7 columns)



**Chapter 2**

**Methodology**

**2.1 Pre-Processing**

Before implementing any Machine Learning model on our training data, we are required to look at the data before we start modeling. However, in data science terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis.

In our case the raw data that we will be working on is text data from a dataset of comments from Wikipedia’s talk page edits.

**2.1.1 Missing value-analysis**

We will first check for any missing values in the comments to check if any comment is null or empty. During which we found that there are no missing values in the comment\_text variable of train dataset.

**2.1.2 General text comments study**

We will do a general text comments study to understand the data text data better, below are the results of that study.

2.1.2.1 All comments

Total no. of comments in test data : 159571

Total no. of clean comments : 143346

Percentage of clean comments : 89.83211235124176

Percentage of offensive comments : 10.167887648758233

Total number of NAN/Null comments : 0

2.1.2.2 Category wise comments

Total no. of toxic comments in train data : 15294

Total no. of severe\_toxic comments in train data : 1595

Total no. of obscene comments in train data : 8449

Total no. of threat comments in train data : 478

Total no. of insult comments in train data : 7877

Total no. of identity\_hate comments in train data : 1405

2.1.2.3 Datatype of train data

id object

comment\_text object

toxic int64

severe\_toxic int64

obscene int64

threat int64

insult int64

identity\_hate int64

dtype: object

2.1.2.4 Average, min and max length of comments

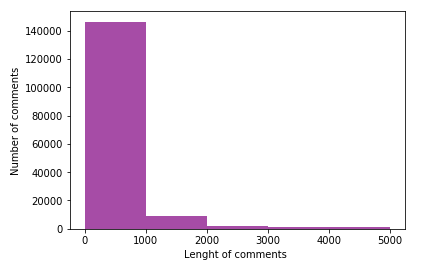
Mean length of comment : 248.48590909375764

Min length of comment : 0

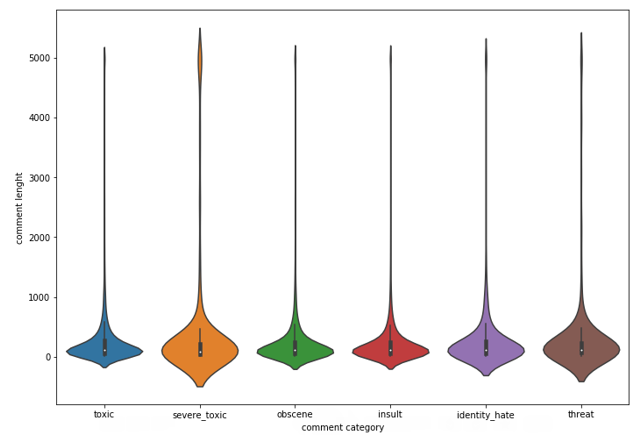
Max length of comment : 5000

**2.1.3 Visualization of dataset**

2.1.3.1 Histogram to understand length distribution of comments



2.1.3.2 Violin plots to see the category wise length distribution of comments



2.1.3.3 Correlation analysis using heatmap

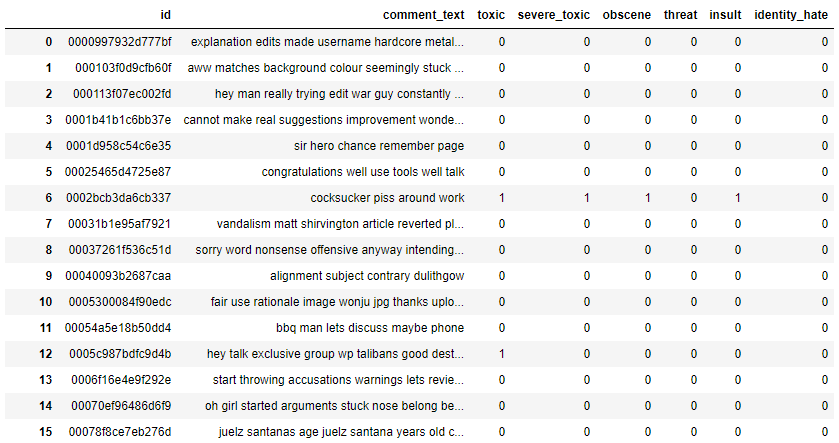


From the above Heatmap we can see there is no highly positive or negative correlation between any two categories.

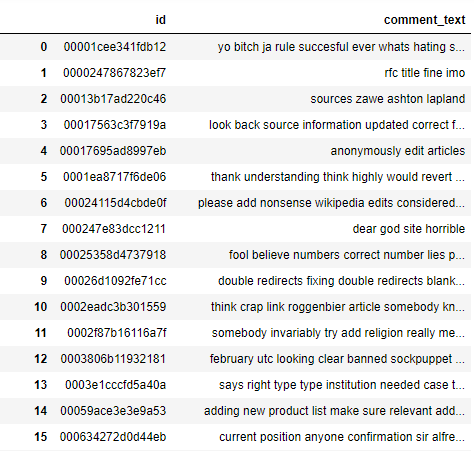
**2.1.4 Data Cleaning**

One of the first steps in working with text data is to clean the data. It is an essential step before the data is ready for analysis. Majority of available text data is highly unstructured and noisy in nature – to achieve better insights or to build better algorithms, it is necessary to play with clean data. For example, social media data is highly unstructured – it is an informal communication – presence of unwanted content like Numbers, Stopwords, punctuations, whitespaces, uppercase etc. are the usual suspects.

2.1.4.1 Train dataset after data cleaning



2.1.4.2 Test dataset after data cleaning



**2.1.5 Word Cloud to category wise visualize frequent comments**

**2.1.5.1 Word cloud for toxic comments**

****

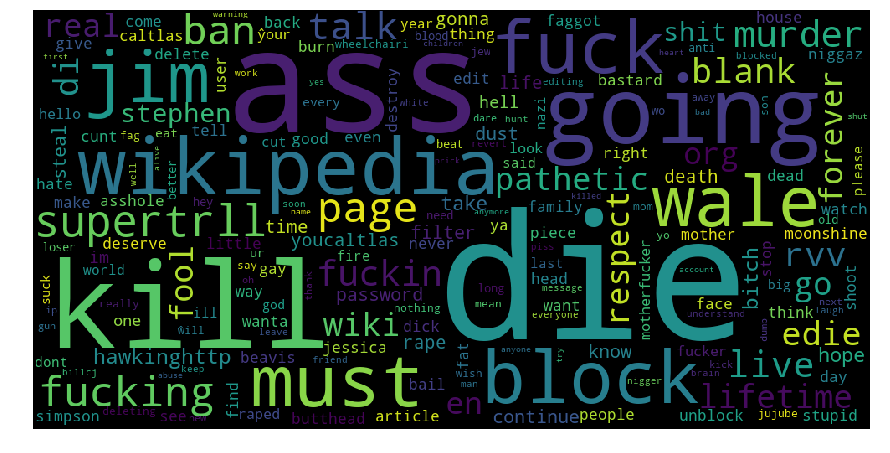
**2.1.5.2 Word cloud for severe\_toxic comments**

****

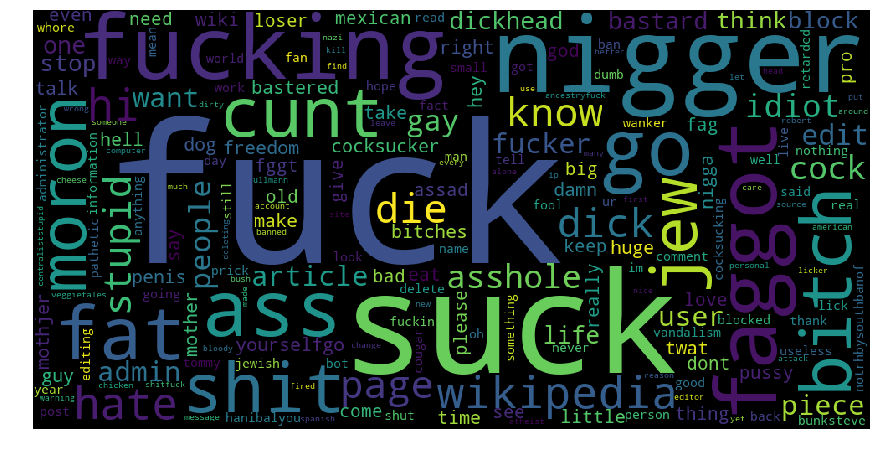
**2.1.5.3 Word cloud for obscene comments**

****

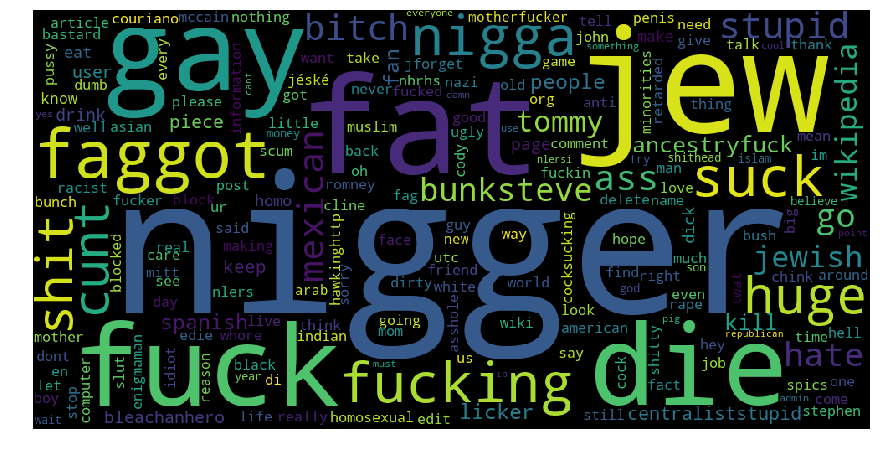
**2.1.5.4 Word cloud for threat comments**

****

**2.1.5.5 Word cloud for insult comments**

****

**2.1.5.6 Word cloud for identity\_hate comments**

****

**2.2 Vectorization of data**

After cleaning the text data, it is needed to transform text into something a machine can understand before feeding it to a machine learning model. That is, transforming text into a meaningful vector (or array) of numbers.

In our model we will be using TF-IDF (term frequency–inverse document frequency) method for vectorization. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. tf-idf is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use tf-idf.

2.2.1 Result of vectorizing train dataset:

<159571x5000 sparse matrix of type '<class 'numpy.float64'>'

with 3049323 stored elements in Compressed Sparse Row format>

2.2.2 Result of vectorizing test dataset:

<153164x5000 sparse matrix of type '<class 'numpy.float64'>'

with 2545080 stored elements in Compressed Sparse Row format>

**2.3 Modeling**

**2.3.1 Model Selection**

In early stage of our analysis process we have come to understand that a particular comment in test data can fall under one or more out of six categories of offensive comments namely toxic, severe\_toxic, obscene, threat, insult and identity\_hate. We have to compute the chances of a comments to fall under each category i.e. probability of a particular comment of being namely toxic, severe\_toxic, obscene, threat, insult, identity\_hate.

Therefore, each comment can be assigned with multiple categories, so these types of problems are known as multi-label classification problem, where we have a set of target labels. Also, the output has to be in the form of probability therefore we will use logistic regression.

Here we will vectorize the data into document-term-matrix and learn the vocabulary of our train data to make a model on top of it and compare the output with the actual values of train data to compute the accuracy of our model.

After that we will implement our trained model on the test dataset and predict the probability of a particular comment under each category.

**2.3.2 Multi-label classification**

A problem where each instance can be assigned with multiple categories, so these types of problems are known as multi-label classificationproblem, where we have a set of target labels.

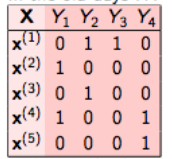
We will use **Problem Transformation method** to solve this problem, here we will transform   
our multi-label problem into multiple single-label problems.

This method can be carried out using binary relevance technique.

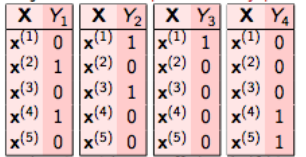
2.3.2.1 Binary Relevance

This is the simplest technique, which basically treats each label as a separate single class classification problem.

For example, let us consider a case as shown below. We have the data set like this, where X is the independent feature and Y’s are the target variable.



In binary relevance, this problem is broken into 4 different single class classification problems as shown in the figure below.



Similarly, we can we can compare this to our problem where X is comment\_text and Y1, Y2, Y3, Y4 etc. are the different categories of offensive comments.

So, we can train a model to individually calculate the probability of each category individually and store the value and iterate the same for other categories of offensive comments to get us to our desired output, which is probability of each comment under each category.

**Chapter 3**

**Conclusion**

**3.1 Model Evaluation**

We will compute the Predictive Performance of our model by comparing Predictions of the models with real values of the target variables and calculating accuracy score.

Our training accuracy for each comment category is as follows:

Training accuracy is 0.9835245752674359 for obscene

Training accuracy is 0.9756346704601714 for insult

Training accuracy is 0.9638217470593028 for toxic

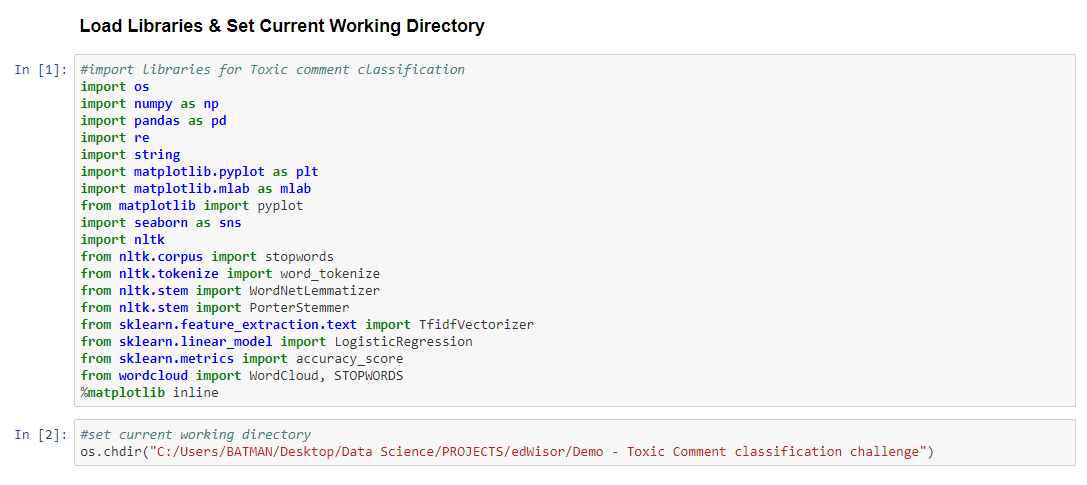
Training accuracy is 0.9919784923325667 for severe\_toxic

Training accuracy is 0.9939400016293688 for identity\_hate

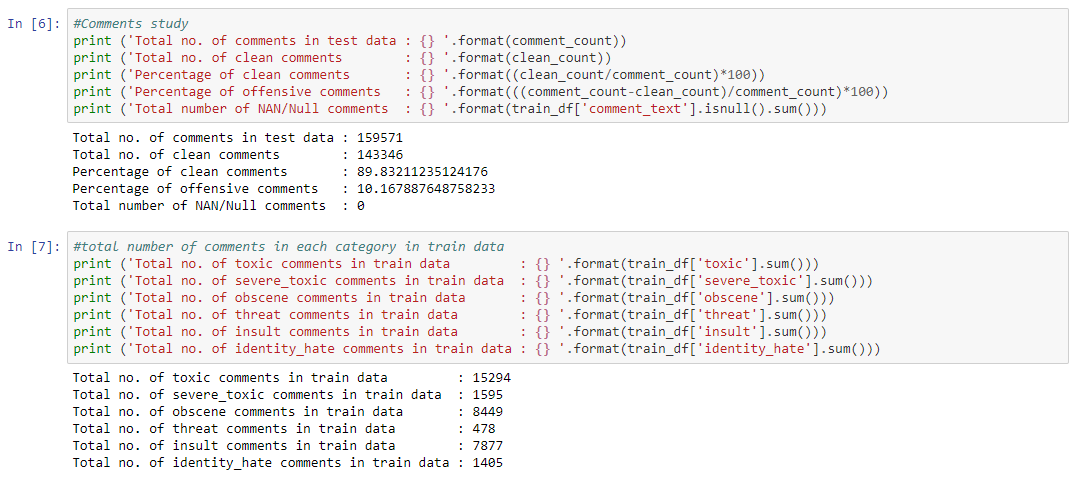
Training accuracy is 0.998038490703198 for threat

**Appendix - Python Code**

**(Along with Output)**

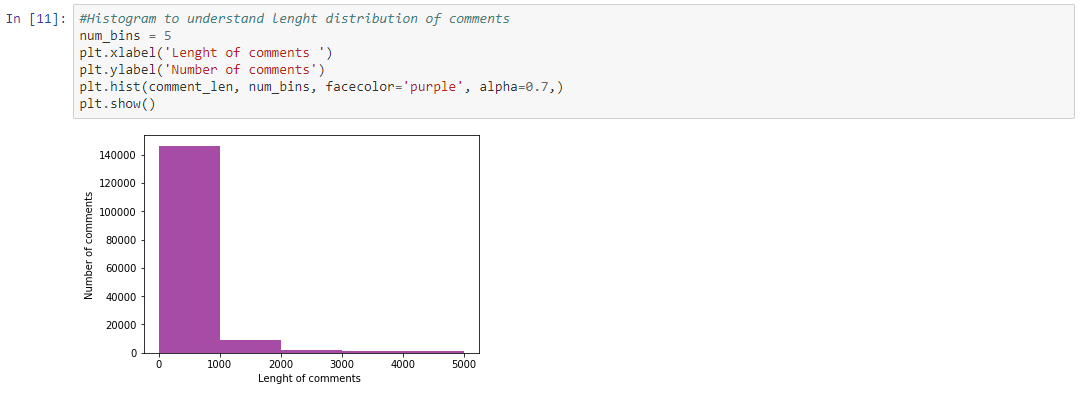




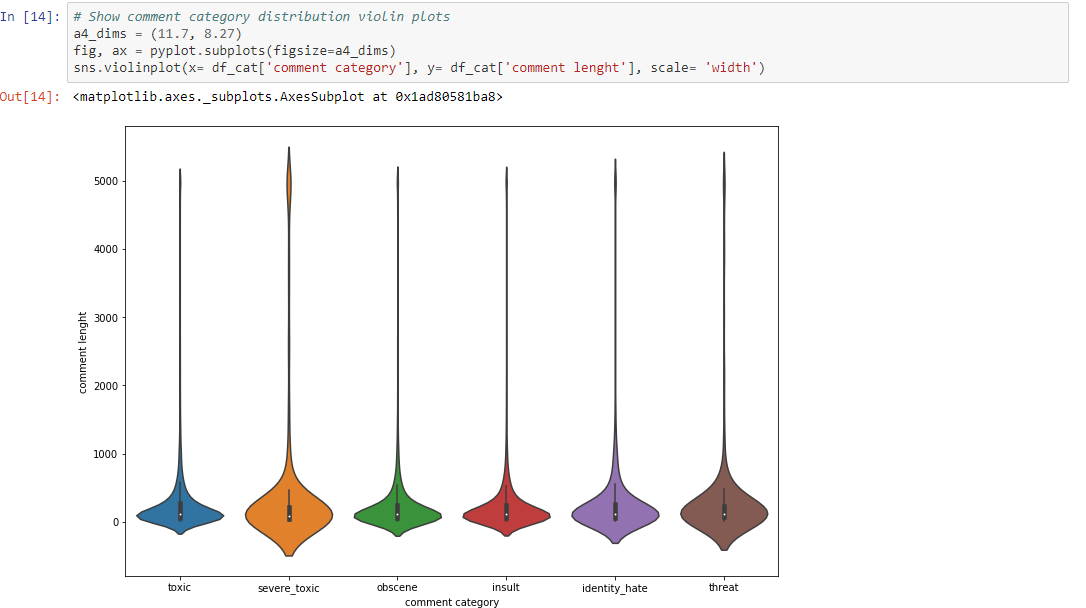




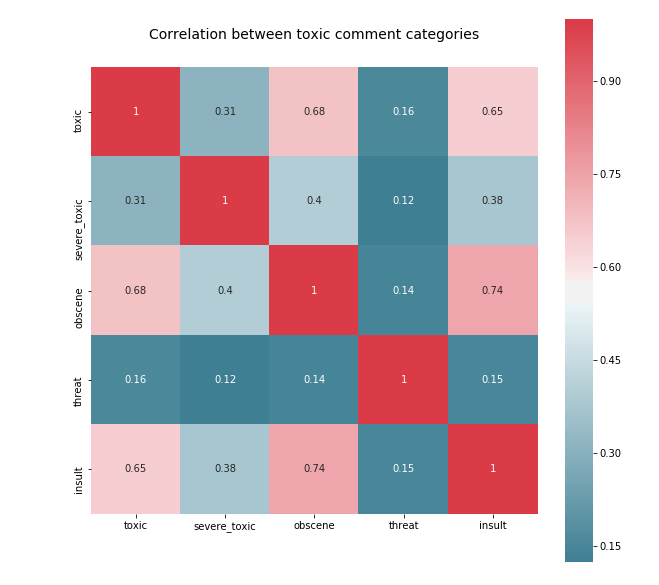


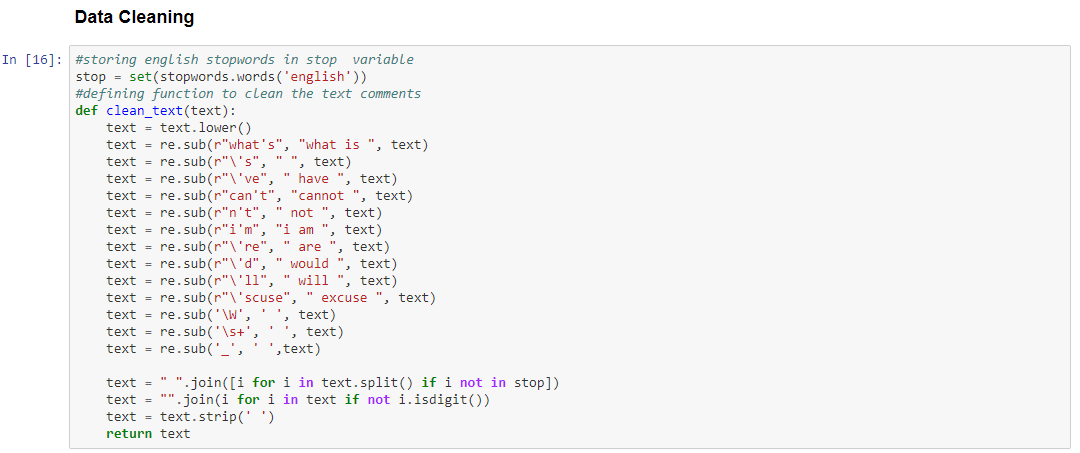






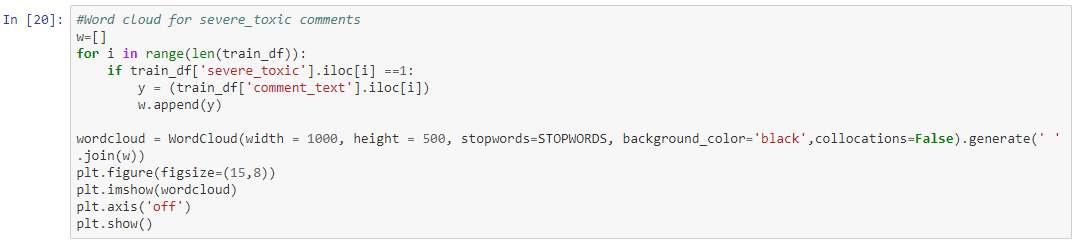




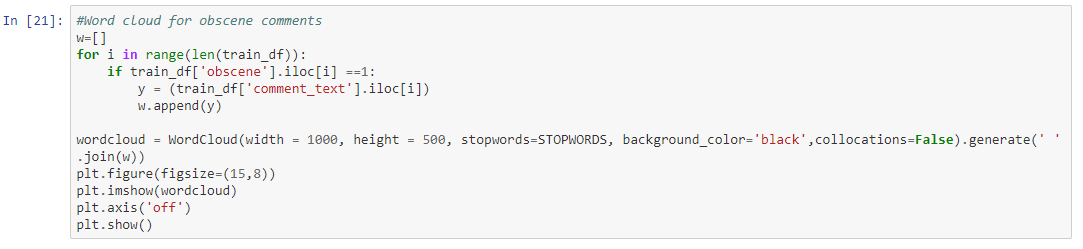




****

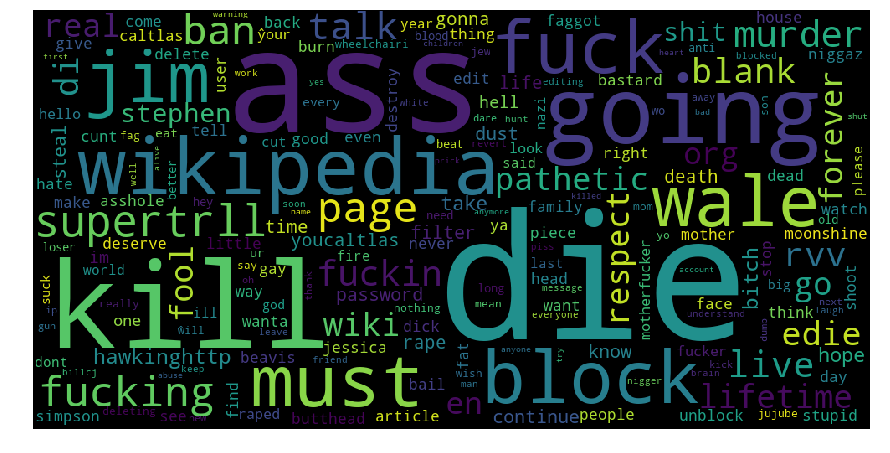


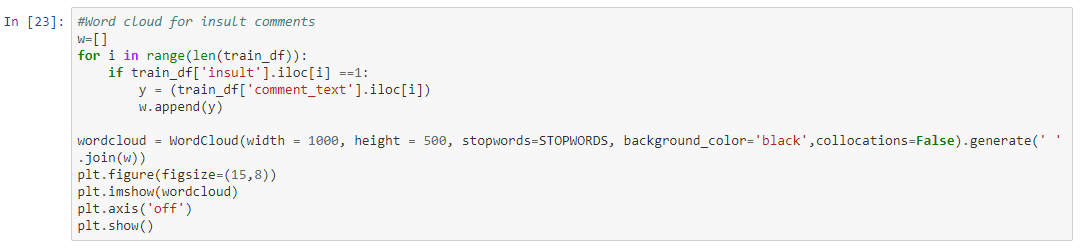
****

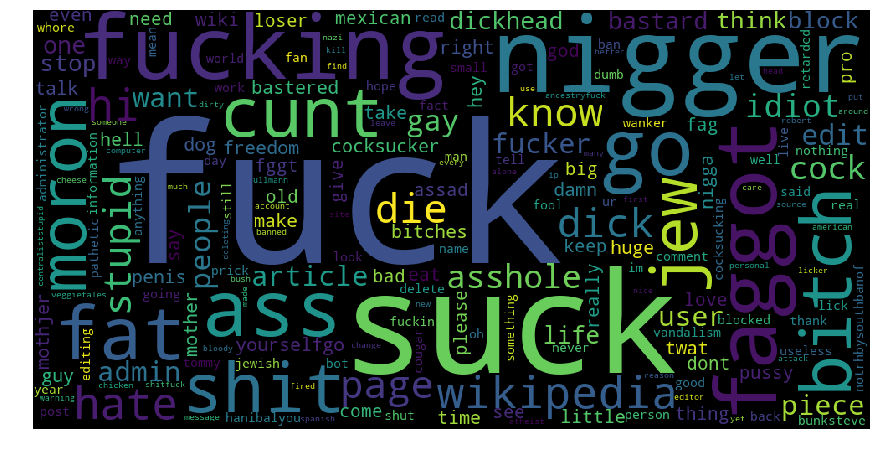


****

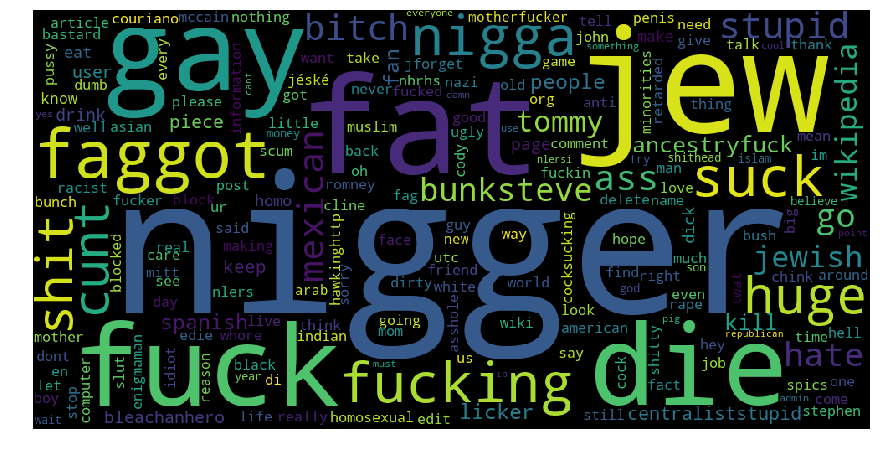


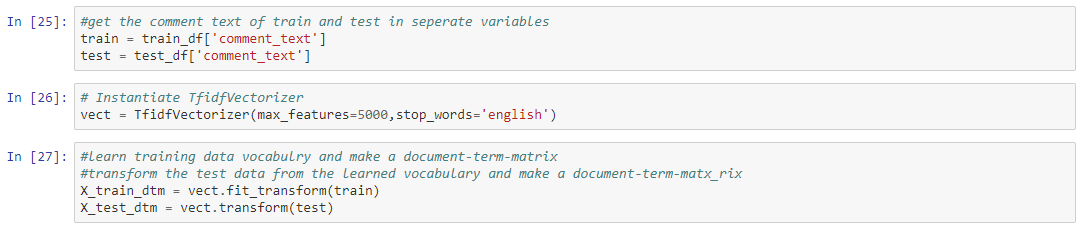
****



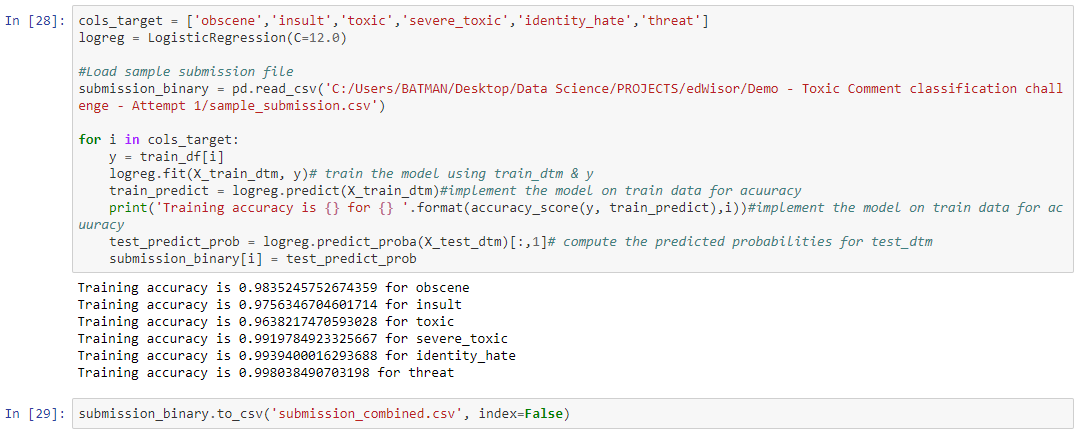
****



****









**References**

[**https://www.analyticsvidhya.com/**](https://www.analyticsvidhya.com/)

[**https://machinelearningmastery.com**](https://machinelearningmastery.com)

[**https://docs.python.org/3/**](https://docs.python.org/3/)