# **Introduction**

This challenge was posted in Kaggle by Jigsaw in collaboration with Google’s Alphabet. The reason we chose this dataset is to classify and remove or block rude comments online. All the comments were drawn from Wikipedia talk page where people can post their opinions and experiences online. By implementation of this project, we can reduce online harassment to provide freedom to post publicly.

# **Goal**

Aim of the project is to correctly classify comments into 6 categories based on level of toxicity.

# **Exploratory Data Analysis and Visualizations**

We have started with Exploratory data analysis to draw insights from the data and accomplish the following tasks –

1. Understand the type of Data, it’s distribution.
2. Visualize data to help detect relevant relationships between variables or class imbalances.
3. Are there any missing Values? If so, plan out the approach to impute them.
4. Normalizing and or Transforming the data if needed.

# **Dataset Info**

We have 6 classification output variables:

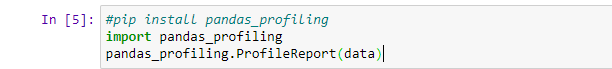
1. Toxic
2. Severe-toxic
3. Obscene
4. Threat
5. Insult
6. Identity-hate

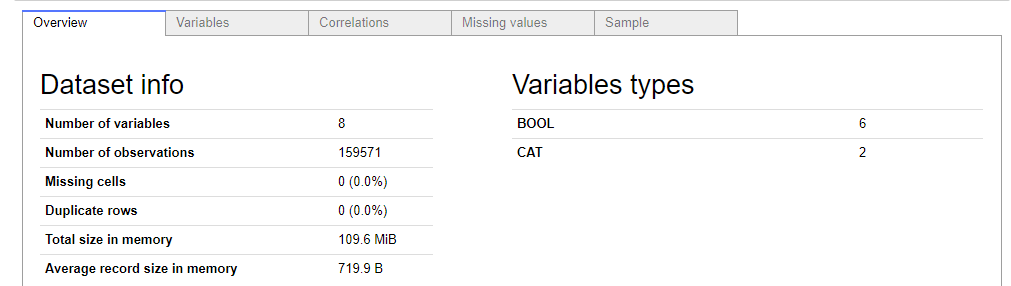
One ID variable and one input variable i.e. comment\_text.

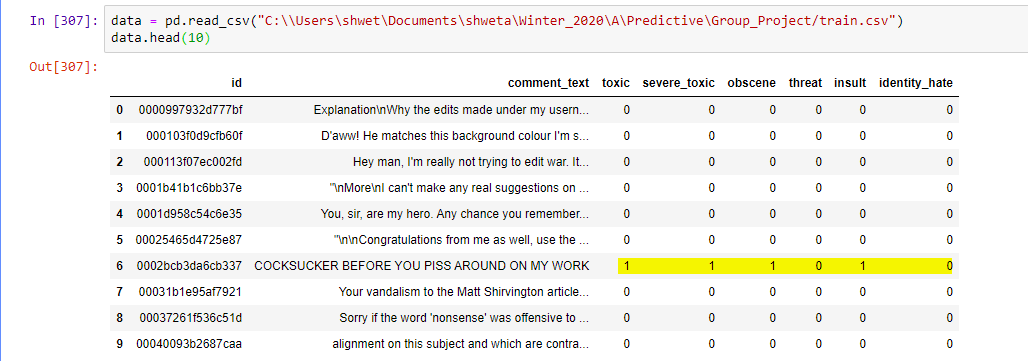
Data description:

* Number of variables: 8
* Number of observations: 159571
* Missing values: 0
* Total size: 109.6 mb

This is evident from the below output the data summary. We have used pandas\_profiling library to summarize the dataset.



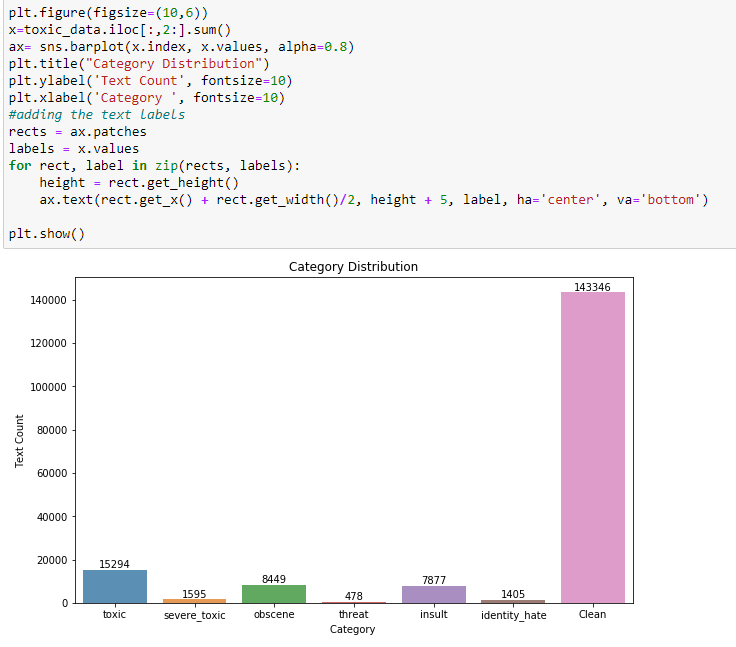




As can be observed from the above output, we are dealing with a multi label classification problem i.e. one comment can be classified in more than one category as highlighted in the above figure.

In the subsequent part we have visualized the class distribution.

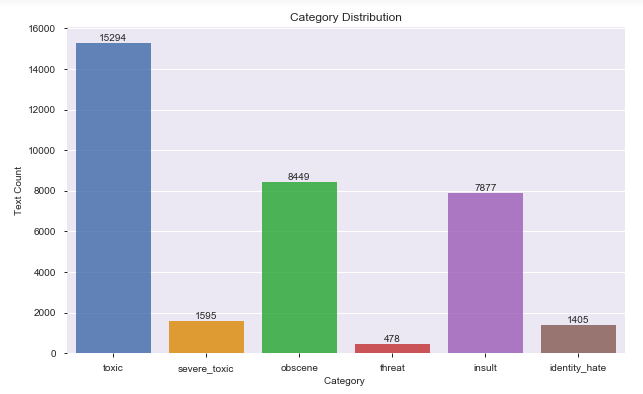
We created new column called “clean” where in the values of all other categories in dataset are zeros. 90% of the comments are clean with no level of toxicity in them.



Since our output variable are the 5 categories, in the following visualization, we would study the distribution of only the toxic categories.

Below Code snippet and bar plot explains about the count of words of comments in each category where in the highest number of comments come under toxic.

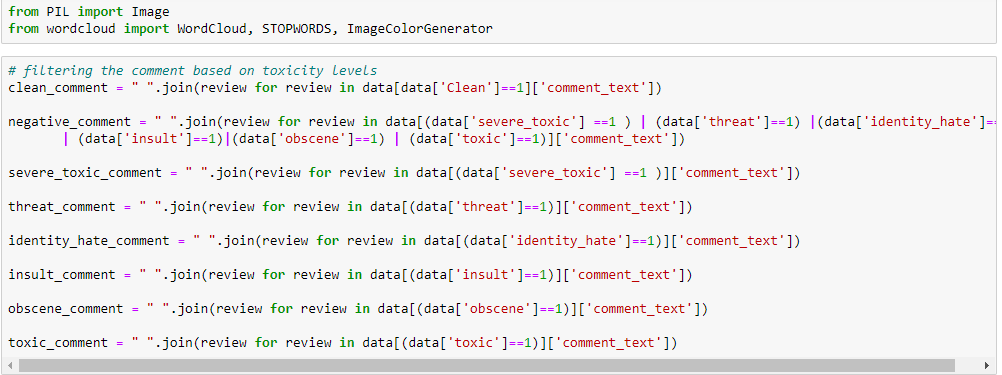




In the next visualization, we have built word clouds for each category to study which words are the most used in the different categories. As can be observed from the below images, there are words which are common among all the categories with almost the same frequency.



Below is the code snippet to visualize the above graph.





After performing EDA, we would proceed with processing of the comments to extract valuable information.

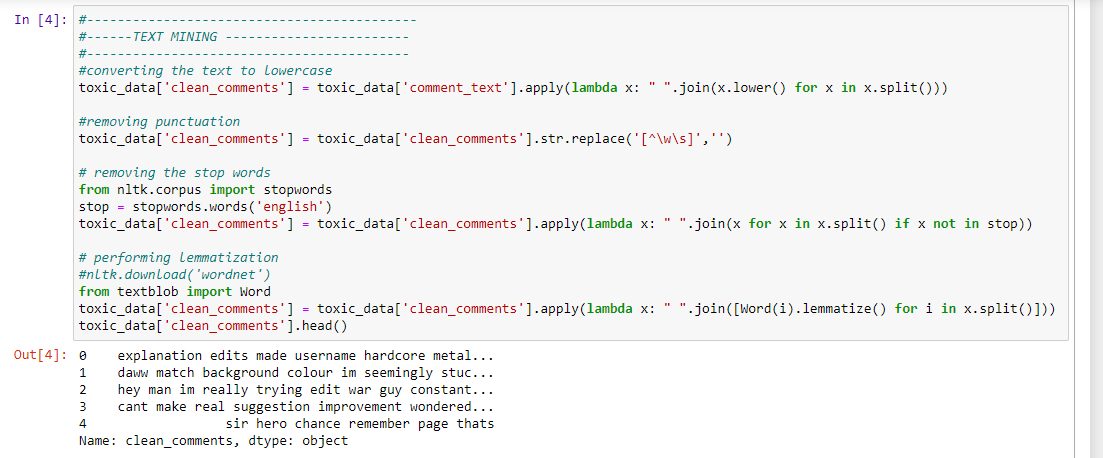
# **Text Mining**

In the next step we have processed the comments to extract valuable information.

Following are the steps we followed-

1. Converting the text to lower case. This would ensure our similar words with different cases are treated as duplicates.
2. Removing Punctuation, Punctuation does not add any value to the text and therefore removing it won’t cause loss of information, on the contrary it would help to reduce the length of the text.
3. Removing Stop words. These are the most common words (such as “the”, “a”, “an”, “in”) used in sentences, which do not add any information. We have used the predefined list of words defined in the library- stopwords of NLTK package to filter out the common words.
4. We performed lemmatization. Lemmatization is the process of converting the word into its dictionary form. For e.g., through Lemmatization, ‘am’, ‘are’, ‘is’ will be converted to ‘be’. We have used the lemmatize () function of the text blob package.

 Below is the code for implementing the above steps.



We have used the split() function to fetch words from the sentence and then perform the cleaning process on it using the lambda function. As can be observed from the above output, we have processed the comment\_text column to consist of only useful words.

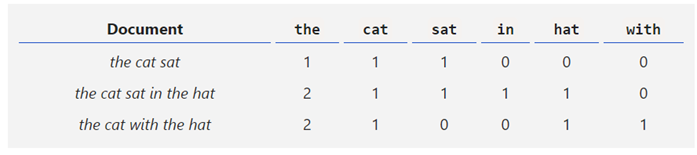
# **Feature Extraction**

We would be performing text feature extraction which is the process of taking out a list of words from the text data and then transforming them into a feature set which would be used as predictors for classification.

There are many ways to extract features using NLP techniques, in this project we have used the following two approach-

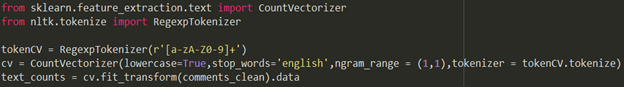
1. Count Vectorization
2. Term Frequency - Inverse Document Frequency (TF-IDF)

**Count Vectorization** - In this method, we count the number of occurrences of each distinct word that appears in the document i.e. each row of comment\_text column in our data set.  Below example, explains the process of count vectorization.



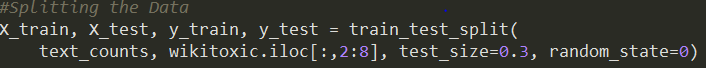
Using the function CountVectorizer() we have converted the comments into a count vector matrix in which every row represents a document from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document(as depicted in the above picture).

Below is the code implementation for the same.



The type and number of words in the token are decided by ngrams. In this case we have used unigram (1,1) i.e. we are considering only one word at a time. The output of the above code is a sparse matrix with 3874854 stored elements in Compressed Sparse Row format which is stored in the variable ‘text\_counts’ and will be used as our set of independent variables for training and testing the data.

We have used Scikit Learn to split the data into training and testing sets. X variables is the ‘text\_counts’ matrix developed in the vectorization and Y variables contains binomial values for all the 6 classes.

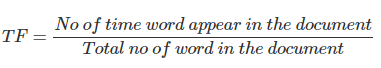


The training set thus created has 111699 rows and test set has 47872 rows.

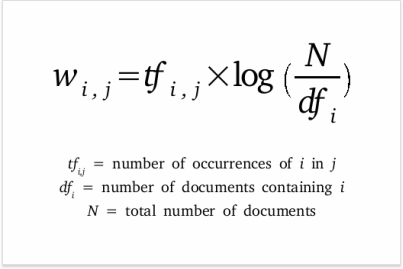
**Term Frequency – Inverse Document Frequency (TF-IDF)-**

It basically tells the importance of the word in the corpus or dataset. This method is a widely used technique in Information Retrieval and Text Mining.

The first term (TF) is the normalized Term Frequency computed as

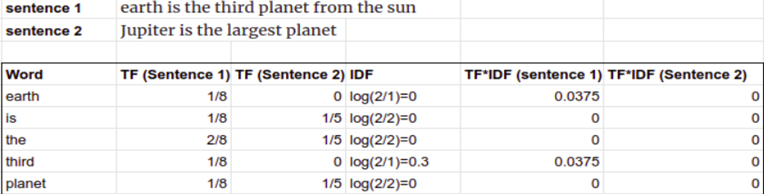


and the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.



<https://www.link-assistant.com/images/news/tf-idf-tool-for-seo/screen-03.png>

The below example explains how the it works.



TF-IDF Vectors can be generated at different levels of input –

1. Word Level TF-IDF: Matrix representing tf-idf scores of every term in different documents.

3. Character Level TF-IDF: Matrix representing tf-idf scores of character level n-grams in the corpus.

We have used TfidfVectorizer library from sklearn to compute the TF-IDF Vectors and have calculated both word and character level to create the features. Below is the implementation of the same. We wanted to use the word and character features both to classify the toxicity and have used hstack function to combine both the features.



We had split the input data into 70% training and 20 %test data and performed the feature extraction.

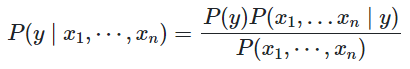
Once our training and test data have been created, we have implemented the classification models for both the techniques (Count Vectorization and TF-IDF) to find the best model and features to predict the levels of toxicity in the comments.

**Classification Models**

**Multinomial Naïve Bayes**

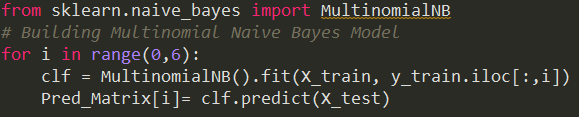
Since we are dealing with word counts as our predictors, we would be applying Multinomial Naive Bayes algorithm which assumes multinomial distribution for all the pairs. It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, Naïve Bayes, predicts the probability of an event happening or not based on the prior knowledge and historical data.

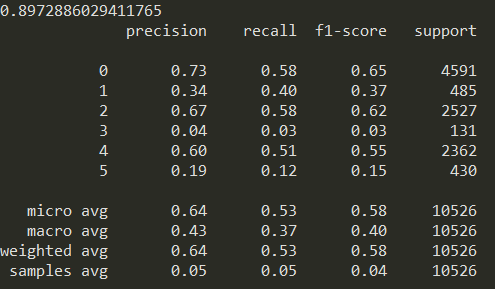
Given a class variable y and dependent feature vector x1 through xn, the probability of features belonging to class y will be -



We have used MultinomialNB() package to calculate the probability of text belonging to the class. Since, our text can belong to multiple classes, we have used loop iterating for each class to build the model. Below is the implementation of the Naïve’s Bayes model.

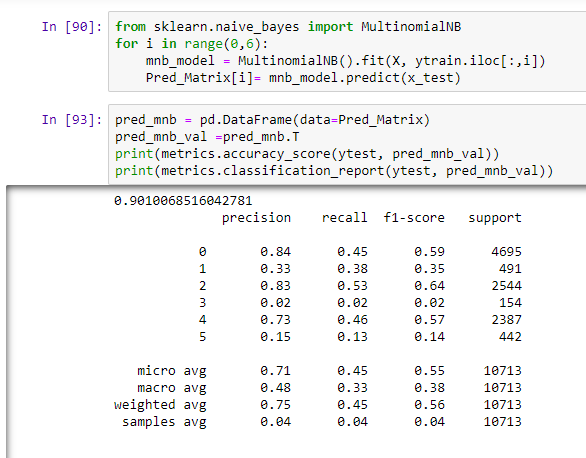
## For Count Vectorization





From the above output we could see, though the accuracy of the model is 90%, the precision and recall are 0.64 and 0.53 which suggests the model is not able to capture the classification accurately.

## For TF-IDF



From the above output we can observed that the accuracy is 91% but there has been an improvement in precision and recall to 0.71 and 0.45 respectively as compared to count vectors features. This suggests that TF -IDF would yield in better classification.

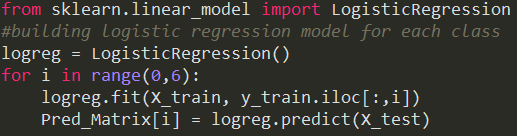
**Logistic Regression**

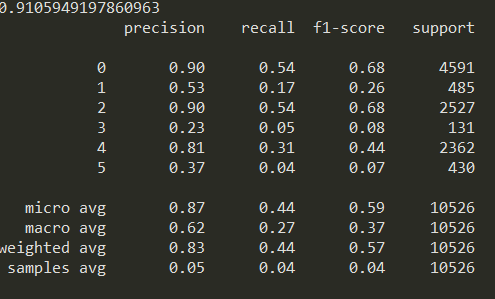
It is a special case of linear regression where the target variable is categorical in nature. method of binary classification of a categorical variable depending on other independent variables. It determines the probability of an event occurring (1) or not occurring (0).

Below is the equation for logistic model in our case.

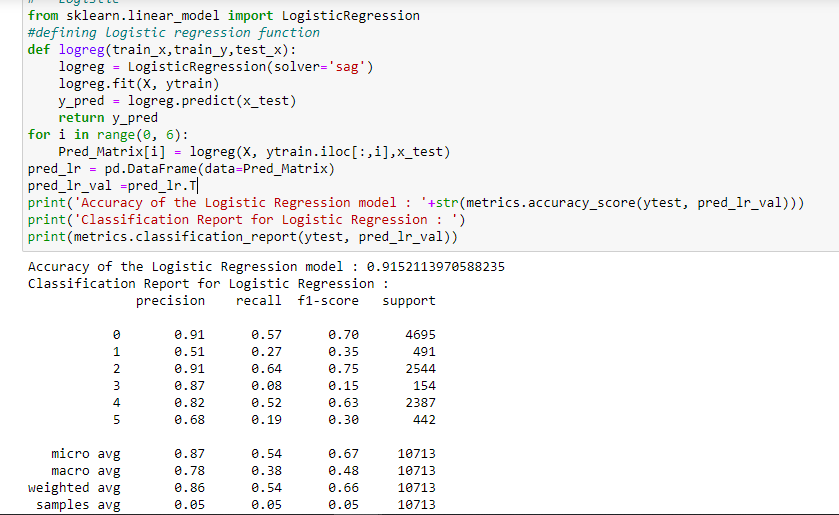
We use the LogisticRegression() package from Scikit Learn and apply the following code to build our model:

## For Count Vectorization





For TF-IDF



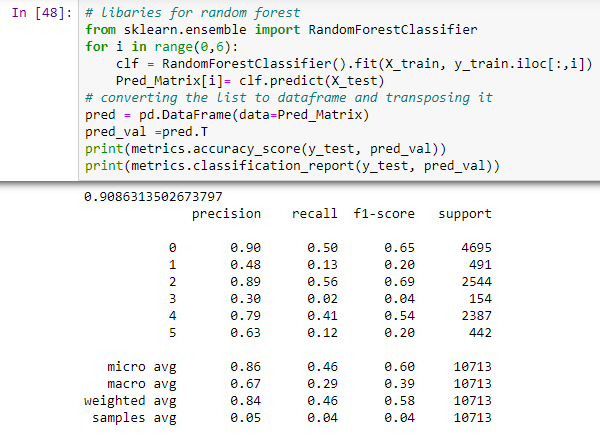
As can be observed from the above outputs for both the features, TF-IDF performs better in terms of precision and recall.

**Random Forest**

Random Forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True.

## Count Vectorization

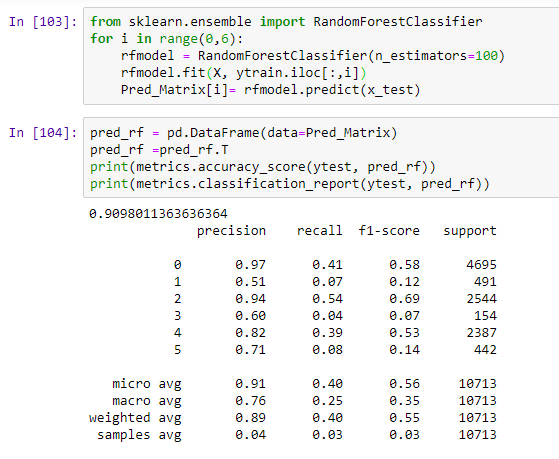
We have used the sklearn.ensemble class and the RandomForestClassifier method using the following code:



The accuracy is 91%, average precision for all the labels is 0.90, recall is 0.50 and F1 Score is 0.65

## TF-IDF

We have used the following code to apply TF-IDF vector in the Random Forest Model:



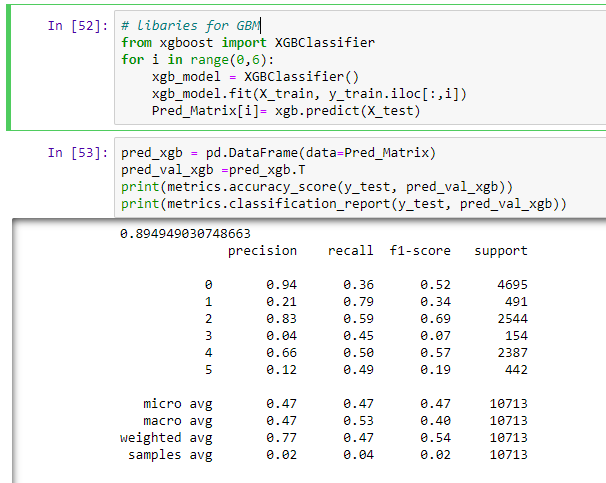
The accuracy is 91%, average precision for all the labels is 0.91, recall is 0.40 and F1 Score is 0.56

**Gradient Boosting**

Gradient Boosting builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n\_classes regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

## Count Vectorization

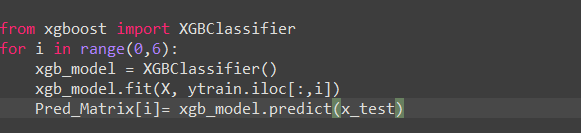
We have used the xgboot class and the XGBClassifier method using the following code:

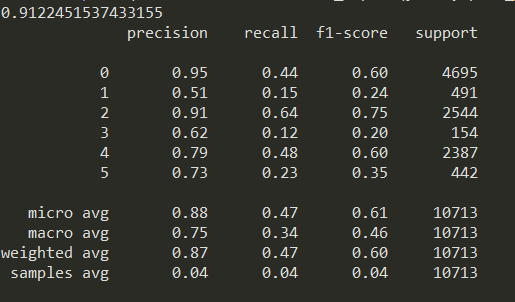


The accuracy is 0.89, which is lower than last week and the random forest model. The precision, recall and F1 Score is also lower from the other models.

## TF-IDF

We have used the following code to apply TF-IDF vector in the Gradient Boosting Model:





# Model Comparison

We will be comparing all the 8 models from count and TF-IDF Vectorization based on 4 model performance parameters:

1. **Accuracy**

Accuracy it is simply a ratio of correctly predicted observation to the total observations. We calculated it by summing the counts along the main diagonal and dividing by the total number of test cases using the following code:



1. **Recall**

Recall is the ratio of correctly predicted positive observations to the all observations in actual class which is calculated using the code below:



1. **Precision**

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations which is calculated using the code below:



1. **F1 Score**

F1 Score is the weighted average of Precision and Recall which is calculated using the code below:



## **Model Comparison**

The four performance parameters for all the eight models are compiled in the table below:

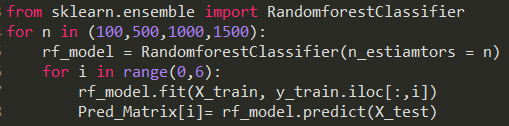
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **TF-IDF** | **TF-IDF** | **TF-IDF** | **TF-IDF** |
| Naïve Bayes | Logit Model | RF Model | XG Boost |
| **Precision** | 0.71 | 0.87 | 0.91 | 0.88 |
| **Recall** | 0.45 | 0.54 | 0.40 | 0.47 |
| **F1 Score** | 0.55 | 0.67 | 0.56 | 0.61 |
| **Accuracy** | 0.90 | 0.92 | 0.91 | 0.91 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **CV** | **CV** | **CV** | **CV** |
| Naïve Bayes | Logit Model | RF Model | XG Boost |
| **Precision** | 0.43 | 0.62 | 0.86 | 0.89 |
| **Recall** | 0.37 | 0.27 | 0.46 | 0.35 |
| **F1 Score** | 0.39 | 0.36 | 0.6 | 0.5 |
| **Accuracy** | 0.89 | 0.91 | 0.91 | 0.9 |

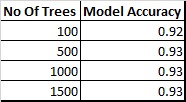
We see that Random forest provides us with the best subset of performance parameters. Hence, we select random forest as our final model.

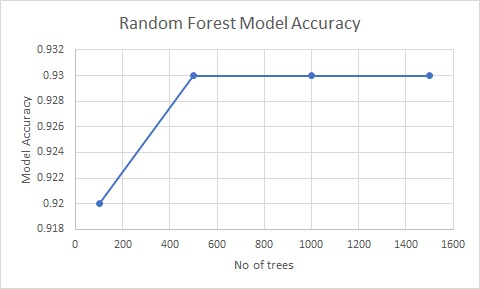
# **Cross Validation of Random Forest Model**

As, we selected Random Forest as our final model, we cross validate our model on different number of estimators to find out the best fit model for our predictions. We test the model on 100, 500, 1000 and 1500 trees to determine the highest accuracy using the following code:

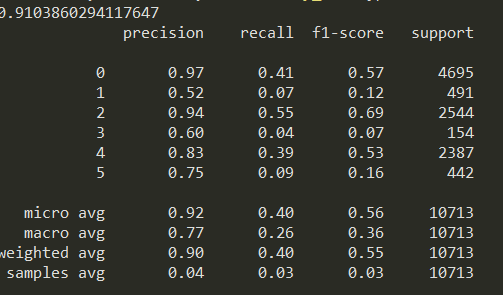


The accuracy from these four random forest models is compiled in the table and graph below:



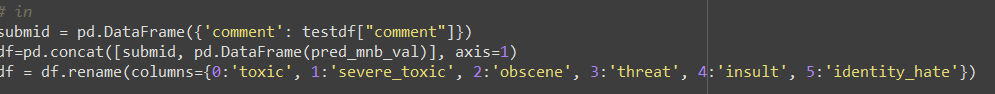


Even though the RF Model with 500 trees provides us the accuracy of 0.93 but the computation cost of running a model with 500 trees was 40% higher than the model with 100 trees. So, we select our final model with 100 trees with following performance parameters:

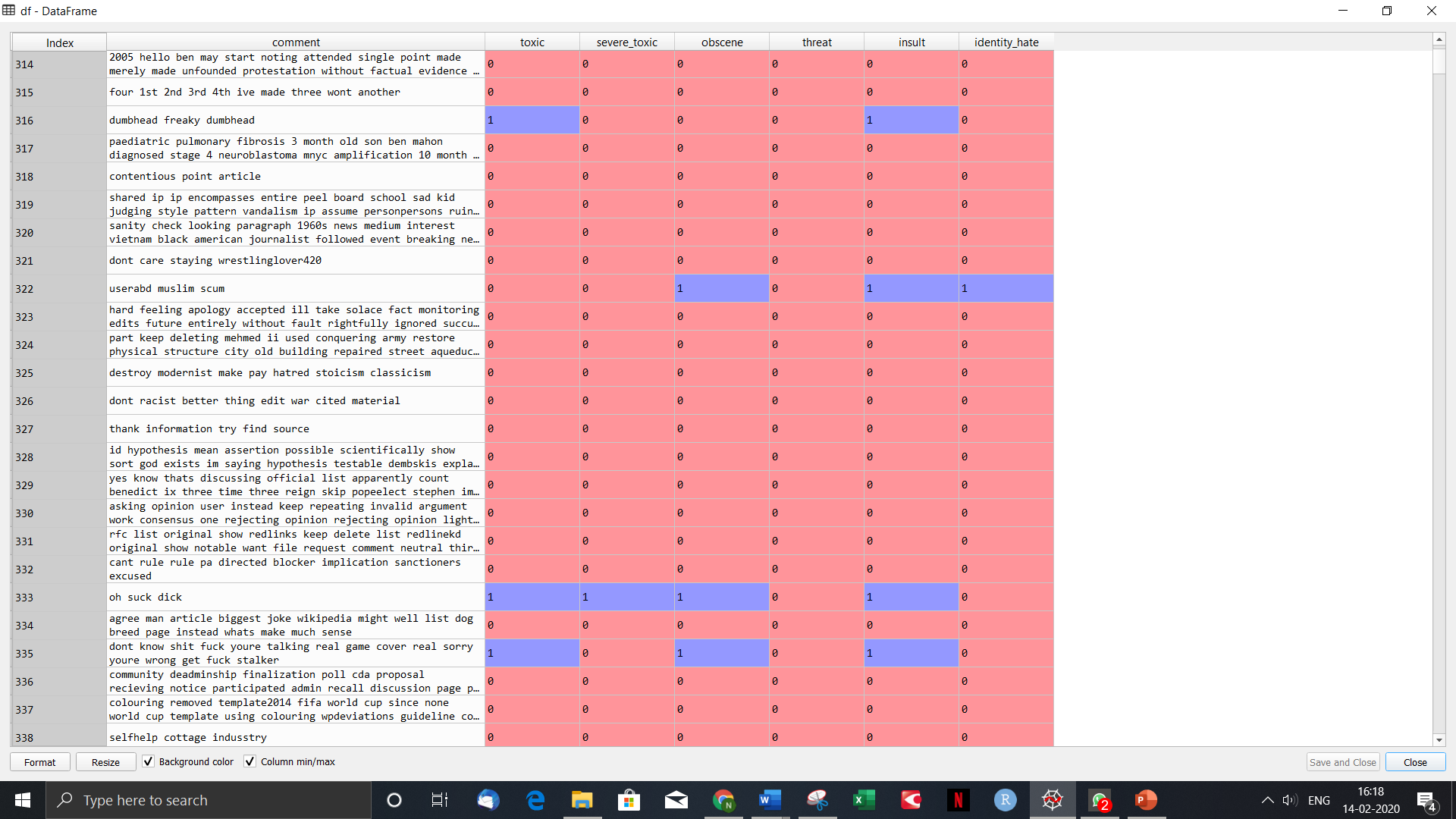


# **Predictions**

We use the following code to compile our predictions from the model in a table:



The output is given below:



**Conclusion**

Through this project we have tries to build a classifier which can predict the toxicity level in a comment and thus can be used to filter out such comments. The best model we had built is Random forest with TF-IDF features. In 2018, Mark Zuckerberg said - "By the end of this year we're gonna have more than 20,000 people working on security and content review," Our toxicity classification model can come to their rescue. While building the models, the major challenge we faced was with random forest and Gradient Boosting algorithms, they were time-consuming and required more computational resources; it took hundreds of megabytes of memory and was slow to evaluate.

The future scope of this project could be to implement Neural Network to improve the efficiency of the model.

References –

1. Unknown, n.d. retrieved from <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data>.
2. [Mohamed Gharibi](https://towardsdatascience.com/@gharibimo?source=post_page-----974d539f21fd----------------------), Devember 2019, retrived from <https://towardsdatascience.com/the-magic-behind-embedding-models-part-1-974d539f21fd>
3. Shivam Bansal, April 2018 retrieved from <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/>