# Toxic comments classifications

**Business problem** :- you’re challenged to build a multi-headed model that’s capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate better than Perspective’s current models. You’ll be using a dataset of comments from Wikipedia’s talk page edits. Improvements to the current model will hopefully help online discussion become more productive and respectful.

**Variables :-**

Index(['id', 'comment\_text', 'toxic', 'severe\_toxic', 'obscene', 'threat',

'insult', 'identity\_hate'],

dtype='object')

**Missing values:-**

id 0

comment\_text 0

toxic 0

severe\_toxic 0

obscene 0

threat 0

insult 0

identity\_hate 0

dtype: int64

In [14]:

**Target variable Multi label classification** :-

data\_columns =['severe\_toxic', 'obscene', 'threat',

'insult', 'identity\_hate']

The method which I followed:-

**Statistical analysis**:- was done on the following

. Numerical data

. Categorical data

**Univar ant and bivariate analysis was done**

**Exploring the data**:-

EDA :-

index 0

0 toxic 2871

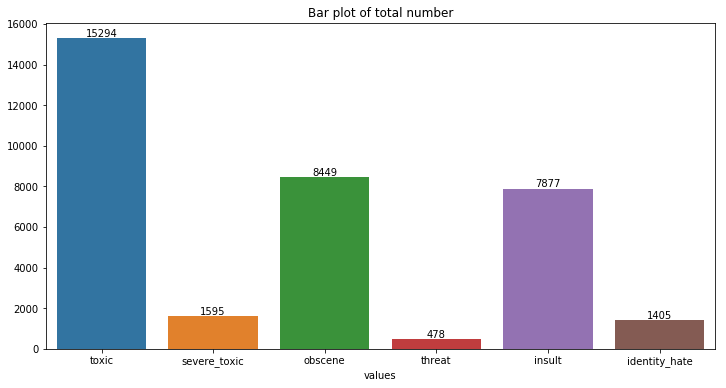
1 severe\_toxic 288

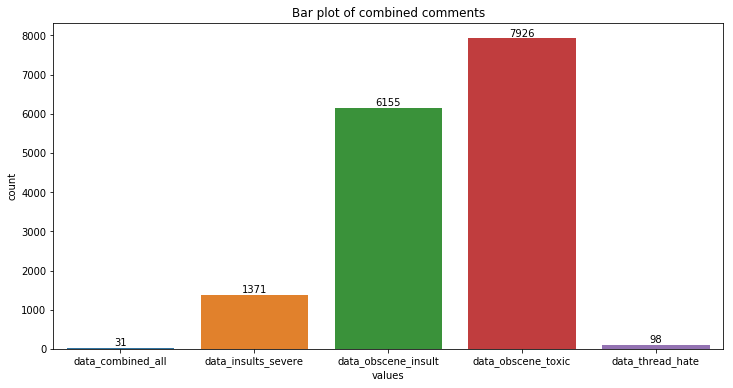
2 obscene 1555

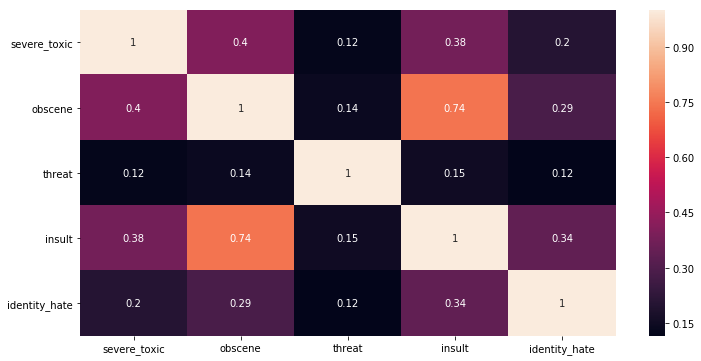
3 threat 96

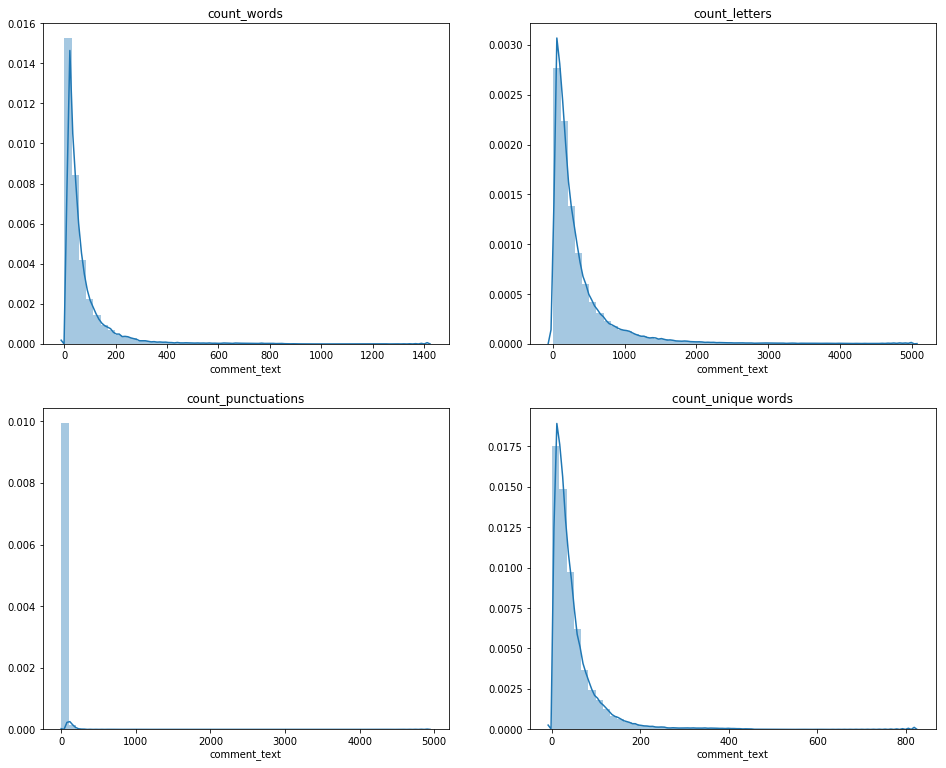
4 insult 1505

5 identity\_hate 266









**Cross tabulation**

toxic severe\_toxic obscene threat insult

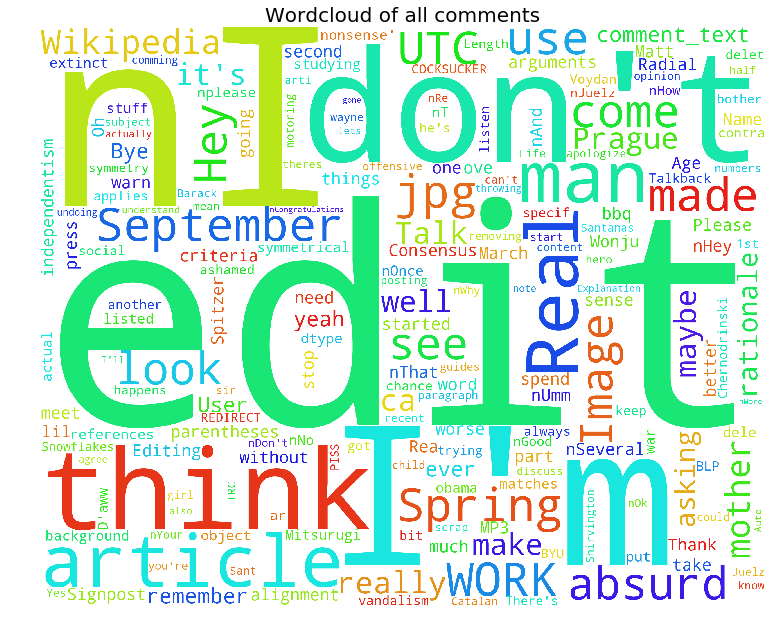
severe\_toxic 0 1 0 1 0 1 0 1 0 1

toxic

0 144277 0 143754 523 144248 29 143744 533 144174 103

1 13699 1595 7368 7926 14845 449 7950 7344 13992 1302

**Word cloud was done :-**

****

**Toxic word cloud:-**

****

**Text analysis:-**

**Text Preprocessing and Wrangling:**

**Breaking Down the sentences:-**

We can think paragraphs as the unit of documents structure, it is useful to see sentences as units of discourse, paragraphs has one complete idea in the same way sentence has complete language structure and we can encode and analyze.

I used sent\_tokenizer from NLTK.

**Tokenization and identifying tokens:-**

Tokens are syntactic units, tokens in the corpus will be sequences of characters that appears in one or more documents and are grouped together to encode some semantic information beyond characters.

**Parts of speech tagging:-**

Parts of speech (e.g. verbs, nouns, prepositions, adjectives) indicate how a word is functioning within the context of a sentence. In English as in many other languages, a single word can function in multiple ways, and we would like to be able to distinguish those uses (for example “building” can be either a noun or a verb). Part-of-speech tagging entails labeling each token with the appropriate tag, which will encode information both about the word’s definition and its use in context.

**Machine learning on Text:-**

**Vectorization:**

**Bag of words:-**

Machine learning algorithms operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features. In order to perform machine learning on text, we need to transform our instances, documents, into vector representations such that we can apply numeric machine learning. The process of encoding documents in a numeric feature space is called feature extraction or more simply, vectorization and is an essential first step towards language aware analysis.

In order to understand vectorization, we must shift from thinking about language as a sequence of words toward thinking about how language might occupy a high-dimensional semantic space. The term space implies a spatial region where each instance is represented by a point; points in space can be close together or far apart. Semantic space is therefore a mapping of meaning to space such that documents that have a similar meaning are closer together and documents that are very different are farther apart. If we can encode similarity as distance we can begin to derive the primary components of documents (ideas) and draw decision regions in semantic space.

The simplest encoding of semantic space is the “bag-of-words” model, whose primary insight is that meaning and similarity is encoded in the specific vocabulary used in each document. For example, a Wikipedia article about baseball and Babe Ruth are probably very similar because the same words will appear in both, whereas they will not share many words in common with an article about quantitative easing. This model, while simple, is extremely effective and is the starting point for more complex models.

**Term Frequency-Inverse Document Frequency:-**

The bag-of-words representations only describe a document in a stand-alone fashion, not taking into account the context of the corpus. A better approach would be to consider the relative frequency or rareness of tokens in the document against their frequency in other documents. The central insight is that tokens that appear frequently in a document have more relevance to that document, particularly if they appear infrequently in the rest of the corpus. For example in a corpus of sports text, in documents that discuss baseball tokens such as umpire, base, dugout will appear more frequently, whereas other tokens that appear frequently throughout the corpus, like run, score, and play, will be less important.

A typical weighting is tf-idf weighting:

w = tf \* idf = tf log2 (N / df)

A term occurring frequently in the document but rarely in the rest of the collection is given high weight.

Experimentally, tf-idf has been found to work well. It was also theoretically proved to work well (Papineni, NAACL 2001)

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and

document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

A: tf = 3/3; idf = log(10000/50) = 5.3; tf-idf = 5.3

B: tf = 2/3; idf = log(10000/1300) = 2.0; tf-idf = 1.3

C: tf = 1/3; idf = log(10000/250) = 3.7; tf-idf = 1.2

**train\_words** = TfidfVectorizer(ngram\_range=(1,1),analyzer='word',strip\_accents='unicode',lowercase=None,stop\_words=Stopwords,max\_features=15000,sublinear\_tf=True)

**Text Normalization: -**

 For text, normalization is intended to reduce the number of features (tokens, in a bag-of-words model) by eliminating or combining different tokens into a single class. Consider case, for example, the tokens friend, Friend, and FRIEND all have the same meaning (possibly) and therefore can be reduced to the single token, friend.

**Stop words:**

* It is typical to exclude high-frequency words (e.g. function words: “a”, “the”, “in”, “to”; pronouns: “I”, “he”, “she”, “it”).
* Stopwords are language dependent
* For efficiency, store strings for stopwords in a hash table to recognize them in constant time.
* How to determine a list of stopwords?
  + For English?

– may use existing lists of stopwords

• E.g. SMART‟s common word list

• WordNet stopword list

• http://www.ranks.nl/resources/stopwords.html

– For Spanish? Bulgarian? Hindi?

**Lemmatization:-**

* Reduce inflectional/variant forms to base form
* Direct impact on VOCABULARY size

– am, are, is make it to ‘be’

– car, cars, car's, cars‘ make it to ‘car’

• “the boy's cars are different colors” make it to “the boy car be different color”

* How to do this?

– Need a list of grammatical rules + a list of irregular words

– Children make to child, spoken make it to speak …

– Practical implementation: use WordNet’s ‘morphstr’ function

**Model:-**

**Logistic Regression (aka logit, MaxEnt) classifier:-** .

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

Important points:

1. GLM does not assume a linear relationship between dependent and independent variables. However, it assumes a linear relationship between link function and independent variables in logit model.
2. The dependent variable need not to be normally distributed.
3. It does not uses OLS (Ordinary Least Square) for parameter estimation. Instead, it uses maximum likelihood estimation (MLE).
4. Errors need to be independent but not normally distributed

## Techniques for Solving a Multi-Label classification problem

Basically, there are three methods to solve a multi-label classification problem, namely:

1. Problem Transformation
2. Adapted Algorithm
3. Ensemble approaches

1. Probelm tranfromation

methods in problem trasormation

1. Binary relevance

2. classifier chain

3. label power

1**. Binary relevance**

In this method we tranfrom multilabel into single label problem

from skmultilearn.problem\_transform import BinaryRelevance

from sklearn.linear\_model import LogisticRegression

classifier = BinaryRelevance(LogisticRegression())

classifier.fit(X\_train, Y\_train)

**result:-** 0.97424816546709325

### Using naive bayes for binary classifcation

**Result :-** 0.94302866267512608

### Chain classifer Logistic reg

In classifier chains, problem would be transformed into different single label problems,

roc\_auc\_score(Y\_test,pred)

**Result :-** 0.95882328411600481

### Using naive bayes for chain classification

Result :- 0.94803078115972106

### 2. Label powerset logistic

label powerset has given a unique class to every possible label combination that is present in the training set.

Result :- 0.97269992711792774

### 3.Ensemble method

Random forest:-

**Random forests** or **random decision forests** are an ensemble learning method for classification, regression and other tasks

LabelPowerset(classifier=RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=300, n\_jobs=1,

oob\_score=False, random\_state=None, verbose=0,

warm\_start=False),

require\_dense=[False, False])

Result :- 0.9651907275737448