Malignant comments classification

Submitted by:

Ankita Ramdas Mhetre

ACKNOWLEDGMENT

I would like to express my deep and sincere gratitude towards Fliprobo technologies for providing me the internship opportunity and a great chance for learning and professional development.

I express my deepest gratitude and special thanks to the subject matter expert (SME), Mr Shubham Yadav who in spite of being extraordinarily busy with his duties, took time out to hear, guide and keep me on correct path ,motivated me for taking part in useful decision & giving necessary advices to do the project and providing invaluable guidance throughout.

His dynamism, vision, sincerity and motivation have deeply inspired me. He has taught me the methodology to carry out the task and to present the project works as clearly as possible. It was a great privilege and honour to work and study under his guidance. I am extremely grateful for what he has offered me.

He was never too busy to spare his valuable time for this work. No words are adequate to express my gratitude towards him. I would also like to thank him for his friendship and empathy.

INTRODUCTION

• Business Problem Framing

The proliferation of social media enables people to express their opinions widely online. However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users. Although researchers have found that hate is a problem across multiple platforms, there is a lack of models for online hate detection.

Online hate, described as abusive language, aggression, cyber bullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour.

There has been a remarkable increase in the cases of cyber bullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

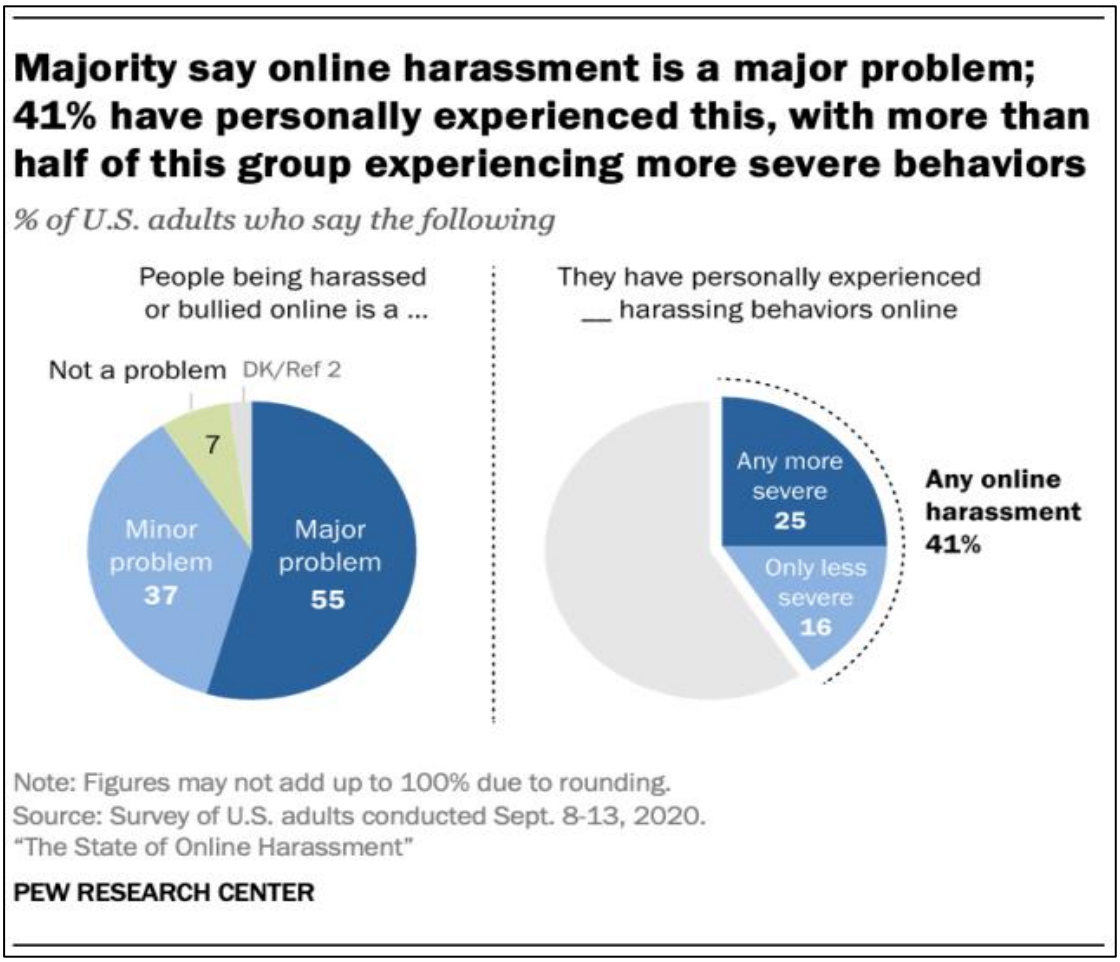
Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate speech, machine learning can be used to fight it. The problem we sought to solve was the tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as inoffensive, but “u are an idiot” is clearly offensive.

Our goal is to build a prototype of online hate and abuse comment classifier which can used to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyber bullying.

This project is more about exploration, feature engineering and classification that can be done on the data. Since the data set is huge and includes many categories of comments, we can do good amount of data exploration and derive some interesting features using the comments text column available. We need to build a model that can differentiate between comments and its categories.

- ## Conceptual Background of the Domain Problem

Due to the penetration of the internet in all domains of life there is an increase in people's participation in all activities. People actively give remarks as an issue of communicating their concern/feedback/opinion in various online forums. Although most of the times these comments are helpful for the creator to extemporize the substance that is being provided to people, but sometimes these may be abusive and create hatred-feeling among the people. Thus as these are openly available to the public which is being viewed from various sections of the society, people in different age groups, different communities and different socio-economic background, it becomes the prime responsibility of the content-creator (the host) to filter out these comments in order to stop the spread of negativity or hatred within people.



Nowadays users leave numerous comments on different social networks, news portals, and forums. Some of the comments are toxic or abusive. Due to numbers of comments, it is unfeasible to manually moderate them, so most of the systems use some kind of automatic discovery of toxicity using machine learning models

Lately there has been many cases in which the growing menace of hate and negativity has been witnessed in the online platforms especially social media as such, many governments around the world has seen the rise in cases related to cyber bullying that has led to spread of hatred and violence.

Few examples of comments are shown below:

A pattern is the use of potentially toxic words within an explanation or self-reproach

Example: *"No matter how upset you may be there is never a reason to refer to another editor as 'an idiot' "*

Comments with Subtle metaphors and comparisons

Example: *"Who are you a sockpuppet for?"*

As sarcasm and irony comments

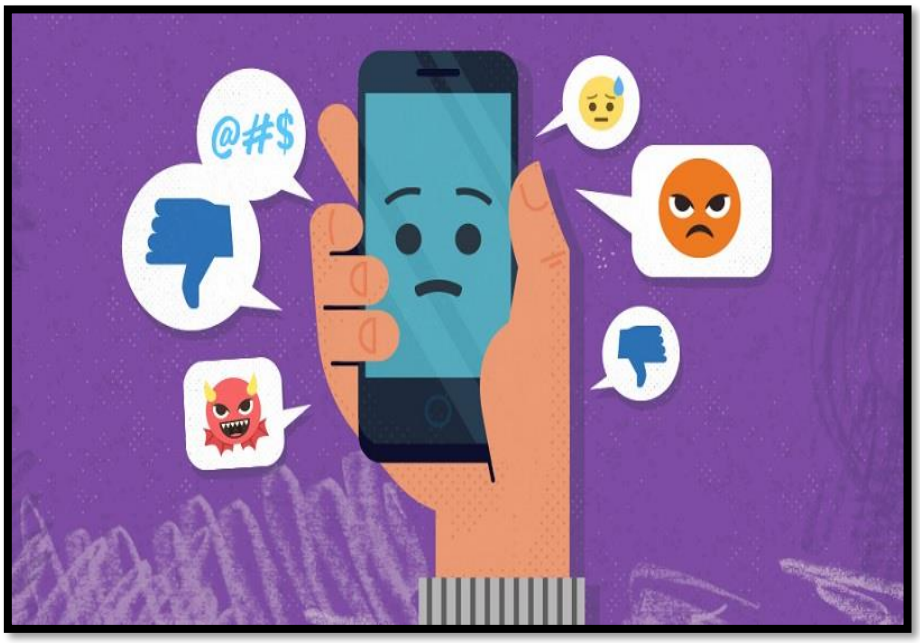
Example: *"hope you're proud of yourself. Another milestone in idiocy."*

Detecting type of comments has been a great challenge for the all the scholars in the field of research and development. This domain has drawn lot of interests not just because of the spread of hate but also because people refrain from participating in online forums which diversely affects all the creators/content-providers to provide a relief to engage in a healthy public interaction which can be accessed by public without any hesitation.

- ## Review of Literature

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.

The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviours, like toxic comments (i.e. Comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they've built a range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).



Toxic comment classification is a complex research problem tackled by several machine learning methods. That is illustrated by many recent works in literature. The research on toxic comment classification became active recently, from 2018. It is due to release of Jigsaw's data set that is mostly used in current related papers. From this year the number of paper is growing and this is an indicator that this research topic is actual and trendy.

The most used data set is the Jigsaw's data set hosted on Kaggle for Toxic Comment Classification Challenge competition. The toxic comment classification research topic is a very active and challenging theme.

Many Machine and Deep Learning Approaches have been attempted for detecting types of toxicity in comments.

- Georgeakopoulos et al. proposed a Deep Learning Approach involving Convolutional Neural Networks (CNNs) for text analytics in toxicity classification, obtaining a Mean Accuracy of 91.2%.

- Khieu et al. applied various Deep Learning approaches involving Long-Short TermMemoryNetworks(LSTMs)for the task of classifying toxicity in online comments, obtaining a Label Accuracy of 92.7%

- Chu et al. implemented a Convolutional Neural Network (CNN) with character level embedding for detecting types of toxicity in online comments, obtaining a Mean Accuracy of 94%.

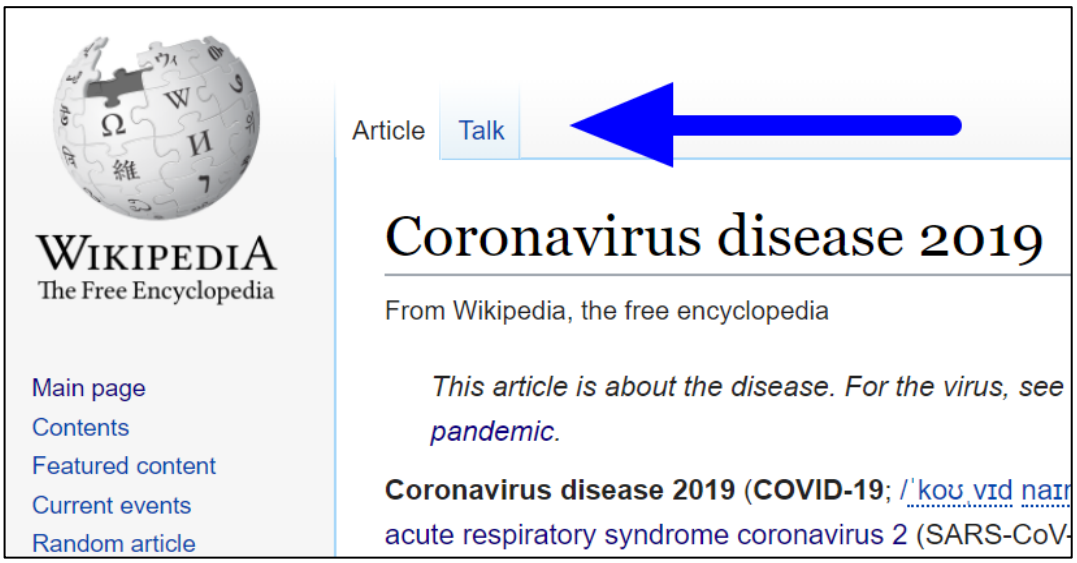
- Kohli et al. proposed a Deep Learning Approach involving Recurrent Neural Network (RNN) Long-Short Term Memory with Custom Embedding for comment toxicity classification, obtaining a Mean Accuracy of 97.78%

Moreover a few datasets have been studied to understand the effect of toxicity in comments .The Twitter dataset by Davidson, Instagram dataset collected by Hosseinmardi et al., Semeval2018- Task 1 with almost 7000 tweets, The Twitter Hate Speech dataset, dataset of tweets made by members of the U.S. House of Representatives, Wikipedia Detox corpus, and live in-game chat conversations from a video game were few of them

Few datasets that have been studied so far are as follows:

Wikipedia Talk Pages dataset:

A dataset published by Google Jigsaw in December 2017 over the course of the 'Toxic Comment Classification Challenge' on Kaggle. It includes 223,549 annotated user comments collected from Wikipedia talk pages and is the largest publicly available dataset for the task.

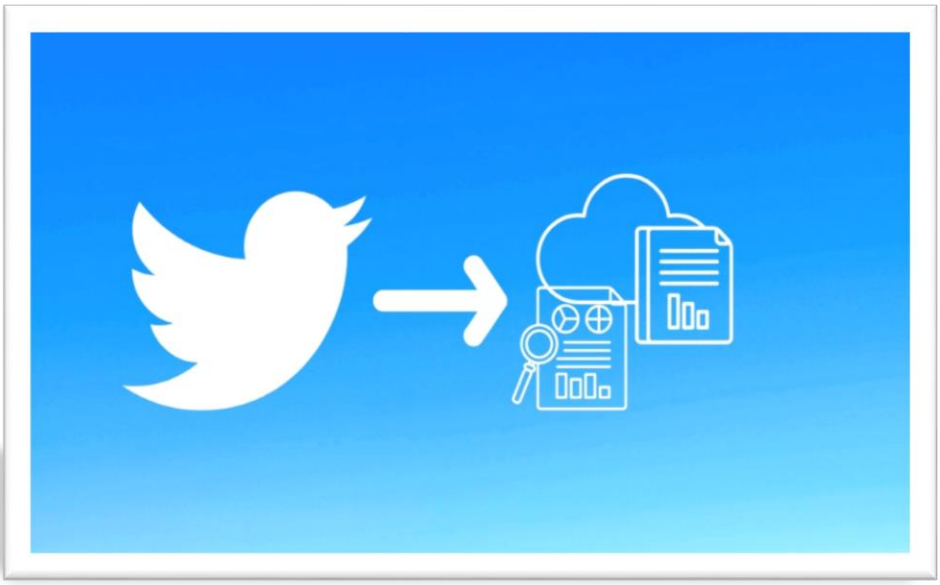


The comments were annotated by human raters with the six labels ‘toxic’, ‘severe toxic, ‘insult’, ‘threat’, ‘obscene’ and ‘identity hate’. Comments could be associated with multiple classes at once, which frames the task as a multi-label classification problem. Jigsaw has not published official definitions for the six classes. But they do state that they defined a toxic comment as “a rude, disrespectful, or unreasonable comment that is likely to make you leave a discussion”. The dataset features an unbalanced class distribution. Comments were collected from the English Wikipedia and are mostly written in English with some outliers, e.g., in Arabic, Chinese or German language.

If the toxic comment can be automatically identified, we could have safer discussions on various social networks, news portals, or online forums. Manual moderation of comments is costly, ineffective, and sometimes infeasible. Automatic or semi-automatic detection of toxic comment is done by using different machine learning methods, mostly different deep neural networks architectures. Recently, there is a significant number of research papers on the toxic comment classification problem, but, to date, there has not been a systematic literature review of this research theme, making it difficult to assess the maturity, trends and research gaps

Twitter dataset:

Additionally a dataset was introduced by Davidson et al. (2017).

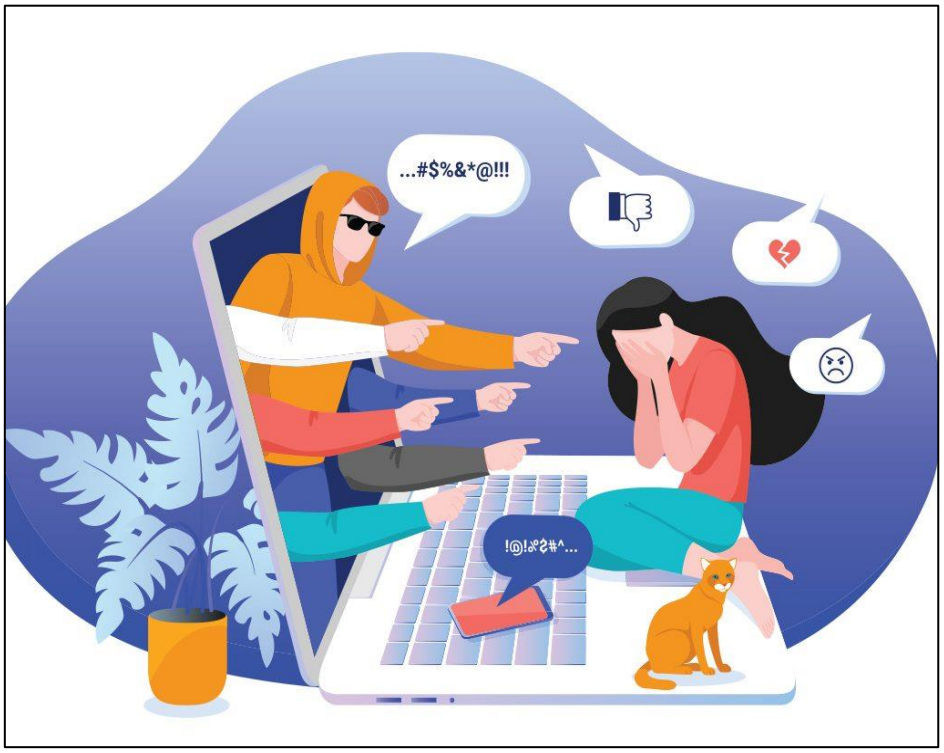


It contains 24,783 Tweets fetched using the Twitter API and annotated by Crowd Flower workers with the labels ‘hate speech’, ‘offensive but not hate speech’ and ‘neither offensive nor hate speech’.

- **Motivation for the Problem Undertaken**

Online forums and social media platforms have provided individuals with the means to put forward their thoughts and freely express their opinion on various issues and incidents. In some cases, these online comments contain explicit language which may hurt the readers. Comments containing explicit language can be classified into myriad categories such as malignant, highly malignant, rude, abuse, loathe, threat, etc. The threat of abuse and harassment means that many people stop expressing themselves and give up on seeking different opinions.

To protect users from being exposed to offensive language on online forums or social media sites, companies have started flagging comments and blocking users who are found guilty of using unpleasant language. Several Machine Learning models can be developed and deployed to filter out the unruly language and protect internet users from becoming victims of online harassment and cyber bullying.



Conversational toxicity is an issue that can lead people both to stop genuinely expressing themselves and to stop seeking others opinion out of fear of abuse or harassment. The goal of this project will be to use machine learning models to identify toxicity in the text, which could be used to help deter users from posting potentially harmful messages ,craft more civil arguments when engaging is discourse with others, and to gauge the toxicity of other user's comments.

Due to the sudden emergence of the crowd using the web, there has been an ascent in the number of mischievous persons too. Now it is the primary task of every online platform provider to keep the conversations constructive and inclusive. The best example can be twitter, a web-based media stage where people share their views. This

platform has already drawn a lot of flak because of the spread of hate speech, insults, threat, defamatory acts which becomes a challenge for many such online providers in regulating them. Thus, there is active research being conducted in the field of Toxic/malignant comment classification.

Online hatred and negativity can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts. Any type of bullying can have physical and psychological effects on a person. Anxiety, fear, depression, low self-esteem and behavioural issues are just of the few challenges people may experience if they are targets. Cyber bullying, however, may be particularly damaging

The inspiration for this toxicity classification challenge comes from the idea of using ML to have better online conversations. Unfortunately, many online discussions devolve into acrimonious arguments, or outright harassment. If conversations are so bad that people leave the discussion, then we have clearly failed to have an online discussion, let alone a good one! This was the basis for working on this project

The goal is that these machine learning models can be used to help online discussion. The project results would help up to create an online interface where we would be able to identify the toxicity level in the given phrase or sentence and classify them into their order of toxicity

Analytical Problem Framing

- ## Mathematical/ Analytical Modeling of the Problem

Exploratory Data Analysis is one of the important steps in the data analysis process. Here, the focus is on making sense of the data in hand — things like formulating the correct questions to ask to your dataset, how to manipulate the data sources to get the required answers, and others.

First let us import the necessary libraries and then we load the data from csv files into a pandas data frame and check its attributes.

```
#import the train dataset
import pandas as pd
import numpy as np
df=pd.read_csv("malignant_train.csv")
df.head()
```

Exploring the characteristics of data:

```
#check the dimensions of the data
df.shape
```

```
(159571, 8)
```

- The dataset has 159571 rows and 8 columns

```
#check the names of columns in dataset
df.columns
```

```
Index(['id', 'comment_text', 'malignant', 'highly_malignant', 'rude', 'threat',
       'abuse', 'loathe'],
      dtype='object')
```

Checking the data type of each column

```
#check the datatype of each feature
df.dtypes
```

```
id                  object
comment_text        object
malignant            int64
highly_malignant     int64
rude                 int64
threat               int64
abuse                int64
loathe               int64
dtype: object
```

Checking if the dataset contains any null values

```
#checking if there are any null values in the dataset
df.isnull().sum()
```

```
id                  0
comment_text        0
malignant           0
highly_malignant    0
rude                0
threat              0
abuse               0
loathe              0
dtype: int64
```

There are null values in our dataset.

```
#alternative way to check null values :yeilds bollean output
df.isnull().any()
```

```
id                  False
comment_text        False
malignant           False
highly_malignant    False
rude                False
threat              False
abuse               False
loathe              False
dtype: bool
```

Calculating number of comments in each category

```
# Calculating number of comments in each category

counts = []
for category in categories:
    counts.append((category, df[category].sum()))
df_stats = pd.DataFrame(counts, columns=['category', 'number of comments'])
df_stats
```

	category	number of comments
0	malignant	15294
1	highly_malignant	1595
2	rude	8449
3	threat	478
4	abuse	7877
5	loathe	1405

- ## Data Sources and their formats

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes ‘Id’, ‘Comments’, ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.
The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

The data set includes following columns:

1. **Malignant**: It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.

2. **Highly Malignant**: It denotes comments that are highly malignant and hurtful.

3. **Rude**: It denotes comments that are very rude and offensive.

4. **Threat**: It contains indication of the comments that are giving any threat to someone.

5. **Abuse**: It is for comments that are abusive in nature.

6. **Loathe**: It describes the comments which are hateful and loathing in nature.

7. **ID**: It includes unique Ids associated with each comment text given.

8. **Comment text:** This column contains the comments extracted from various social media platforms.

Sample instances of dataset are shown below:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

- ## Data Preprocessing Done

Text pre-processing is an important step for any natural language processing task. It transforms text from its raw format into something that is more process able for computer algorithms. The goal of our pre-processing is to achieve normalization and noise removal from the dataset before we experiment with different embedding and models.

We first convert the comments to lower-case and then use custom made functions to remove html-tags, punctuation and non-alphabetic characters from the comments.

```
#import libraries for text cleaning and processing
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
import re

#convert all text into lowercase
data['comment_text'] = data['comment_text'].str.lower()
#check if the text is lower cased
data['comment_text'][0:2]
```

Fig: Convert the entire text into lower case

```
#clean all html tags
def cleanHtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(sentence))
    return cleantext
#apply the function
data['comment_text'] = data['comment_text'].apply(cleanHtml)
data['comment_text'][0:2]
```

Fig: Cleaning the text from any html tags

```
#remove punctuations
def cleanPunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    cleaned = cleaned.strip()
    cleaned = cleaned.replace("\n"," ")
    return cleaned
#call the function
data['comment_text'] = data['comment_text'].apply(cleanPunc)
data['comment_text'][0:2]
```

Fig: Removing any punctuations from the text

```
# any alphabets from small a to small z or capital A to capital Z.
#The plus sign specifies that string should have at least one character.
def keepAlpha(sentence):
    alpha_sent = ""
    for word in sentence.split():
        alpha_word = re.sub('[^a-z A-Z]+', ' ', word)
        alpha_sent += alpha_word
        alpha_sent += " "
    alpha_sent = alpha_sent.strip()
    return alpha_sent
#apply the function
data['comment_text'] = data['comment_text'].apply(keepAlpha)
data['comment_text'][0:2]
```

Fig: keep only alpha numeric data

Stop word removal: Next we remove all the stop-words present in the comments using the default set of stop-words that can be downloaded from NLTK library. Stop words are basically a set of commonly used words in any language, not just English. The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead.

```
#removing stopwords
stop_words = set(stopwords.words('english'))
re_stop_words = re.compile(r"\b(" + "|".join(stop_words) + ")\\W", re.I)
def removeStopWords(sentence):
    global re_stop_words
    return re_stop_words.sub(" ", sentence)
#apply the remove_stopwords function
data['comment_text'] = data['comment_text'].apply(removeStopWords)
data.head()
```

Fig: removing stop words to reduce the vocabulary

Stemming. Stemming condenses the dictionary by identifying root words. There exist different kinds of stemming which basically transform words with roughly the same semantics to one standard form. For example, for amusing, amusement, and amused, the stem would be amus.

```
#stemming to get the root words
stemmer = SnowballStemmer("english")
def stemming(sentence):
    stemSentence = ""
    for word in sentence.split():
        stem = stemmer.stem(word)
        stemSentence += stem
        stemSentence += " "
    stemSentence = stemSentence.strip()
    return stemSentence

data['comment_text'] = data['comment_text'].apply(stemming)
data.head()
```

Fig: stemming

Transforming the text into vectors: When attempting to train a machine learning model on text, several steps are required to transform the text into the format necessary for training. TF-IDF is a numerical statistic that also reflects how important a word is to a document or paragraph. The scheme is split into two calculations, TF and IDF.

$$tf_{t,d} = \frac{n_{t,d}}{Number\ of\ terms\ in\ the\ document}$$

$$idf_t = \log \frac{number\ of\ documents}{number\ of\ documents\ with\ term\ 't'}$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(ngram_range=(1,3))
vectorizer.fit(train_text)
vectorizer.fit(test_text)

#transforms the words into vectors
x_train = vectorizer.transform(train_text)
#drop id and comments column and assign the remaining to y (labels)
y_train = train.drop(labels = ['id','comment_text'], axis=1)

x_test = vectorizer.transform(test_text)
y_test = test.drop(labels = ['id','comment_text'], axis=1)
```

Fig: Converting text into vectors (TF-IDF)

- ## Data Inputs- Logic- Output Relationships

The relation between all labels is studied using correlation function (df.corr ())

Maximum correlation can be seen between rude and abuse which has a correlation co-efficient of 0.74.

The second highest correlated variables are rude and malignant with correlation co-efficient of 0.67 and abuse and malignant with correlation coefficient of 0.64.

Threat and loathe have comparatively very less correlation with other labels.

```
df.corr()
```

	malignant	highly_malignant	rude	threat	abuse	loathe
malignant	1.000000	0.308619	0.676515	0.157058	0.647518	0.266009
highly_malignant	0.308619	1.000000	0.403014	0.123601	0.375807	0.201600
rude	0.676515	0.403014	1.000000	0.141179	0.741272	0.286867
threat	0.157058	0.123601	0.141179	1.000000	0.150022	0.115128
abuse	0.647518	0.375807	0.741272	0.150022	1.000000	0.337736
loathe	0.266009	0.201600	0.286867	0.115128	0.337736	1.000000

Fig: correlation between labels

Checking the distribution of variables using df.skew() function. But as the skewness is in label we are not supposed to treat them, this is just to get an overview of how variables are working.

```
df.skew()

malignant            2.745854
highly_malignant     9.851722
rude                 3.992817
threat              18.189001
abuse                4.160540
loathe              10.515923
dtype: float64
```

Fig: skewness of the variables

- ## Hardware and Software Requirements and Tools Used

Open source web-application used for programming:

1. **Jupyter Notebook**

Python Libraries / Packages used were:

1. **Pandas**: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

   We have used pandas to import the csv file using pd.read_csv all data analysis have been done using the pandas and numpy libraries. The data characteristics have been studied using pandas functions like pd.shape(), pd.dtypes, pd.columns etc.

2. **NumPy:** NumPy is an open-source numerical Python library. NumPy contains a multi-dimensional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic.

3. **Matplotlib**: library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

   The Matplotlib libraries pyplot function is used for making plots ,plt.show() that has been used is a part of matplotlib library.

4. **Seaborn**: Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

   All the visualizations made are built using the saeborn library. Alias used for seaborn is sns.

5. **NLTK**: NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum. NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

6. **Re**: regular expression library, a regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world. The Python module re provides full support for Perl-like regular expressions in Python.

7. **Wordcloud**: Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analysing data from social network websites.

8. **Sklearn:** Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
   All the Machine Learning model have been imported from the sklearn package
   The evaluation metrics like classification reports and accuracy score etc., functions are also imported from the same.

9. **Scikit multi-learn:** Scikit-multilearn is a BSD-licensed library for multi-label classification that is built on top of the well-known scikit-learn ecosystem.Scikit-multilearn provides many native Python multi-label classifiers classifiers. Scikit-multilearn is faster and takes much less memory than the standard stack of MULAN, MEKA & WEKA. Scikit-multilearn is compatible with the Scipy and scikit-learn stack.

Model/s Development and Evaluation

- ## Identification of possible problem-solving approaches (methods)

1. As the aim was to find out whether the data belongs to zero, one or more than one from the six in the list, the initial agenda before working on the problem was to audibly differentiate between multi-label and multi-class classification.
2. In multi-class classification, we undertake one basic assumption that our data can belong to only one label out of all that are available to us. Let us say, for a given picture of a vegetable may be a potato, cabbage, or onion only and not a combination of the above.
3. Whereas, in multi-label classification, data can belong concurrently to more than one label. For example, in this project a comment may be belonging to more than one classification concurrently, like it may be malignant, hateful, obscene and abusive at the same time it might concomitantly belong to non-toxic category and thus does not have an affinity to any of the six labels which are used for classification.

   Let us understand multi-label and multi-class classification with a detailed example. Take a look at the image below. Suppose I ask you that do this image contain a house? The option will be YES or NO.

Consider another case, like what all things (or labels) are relevant to this picture?

House	Tree	Beach	Cloud	Mountain	Animal
Yes	Yes	no	Yes	no	no

These types of problems, where we have a set of target variables, are known as multi-label classification problems.

In short, when there are multiple categories but each instance is assigned only one, such problems are known as multi-class classification problem. For example a movie can be classified as U/A

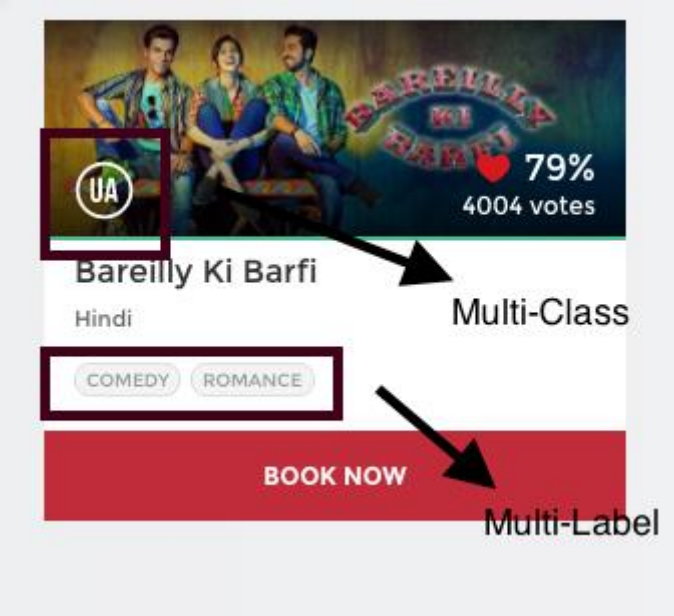


Fig: example of multi class and multi-label

If, each instance can be assigned with multiple categories, these types of problems are known as multi-label classification problem, where we have a set of target labels. For example a movie can fall into romantic and comedy genre at the same time

4. The length of the remarks is pretty big so we performed a couple of visualizations in order to make the data more understandable.
5. We dealt with the number of comments belonging to various categories (which can be perceived from a decisive visualization). Malignant comments were highest in number, followed by rude, abuse, highly malignant loathe and abuse in decreasing order.
6. There were comments which were both malignant and rude at the same, few were malignant, rude and abuse likewise there were many such combinations for the type of comments.
7. The succeeding step was to perform pre-processing of the data. In pre-processing we started off with removal of punctuation and special character from the comments.
8. We also removed the stop words which are useless as they do not add any value to the dataset i.e. those comments were meaningless.
9. Further we performed stemming for the words.
10. Lastly, we applied tfidf vectorizer and afterwards split the information into preparing and testing for the further use.
11. The next steps involved splitting the data into train and test.The test size used is 0.30 which states 80% of the data is used for training and 20% of the data is used for testing.
12. After the train test split we have used various multi label classification algorithms to train and test the data.
13. Respective evaluation metrics are also studied we shall see the details further.

- ## Testing of Identified Approaches (Algorithms)

After identification of the problem as multi-label classification we need to understand how we can build a model for it.

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

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

After the final processing of the data, we applied several supervised learning algorithms including **Binary relevance, classifiers chain, label power set and adaptive algorithm.** Their respective performances on the classification of the data were subsequently compared to identify the best algorithm.

The multi label classification tree is shown in the below image

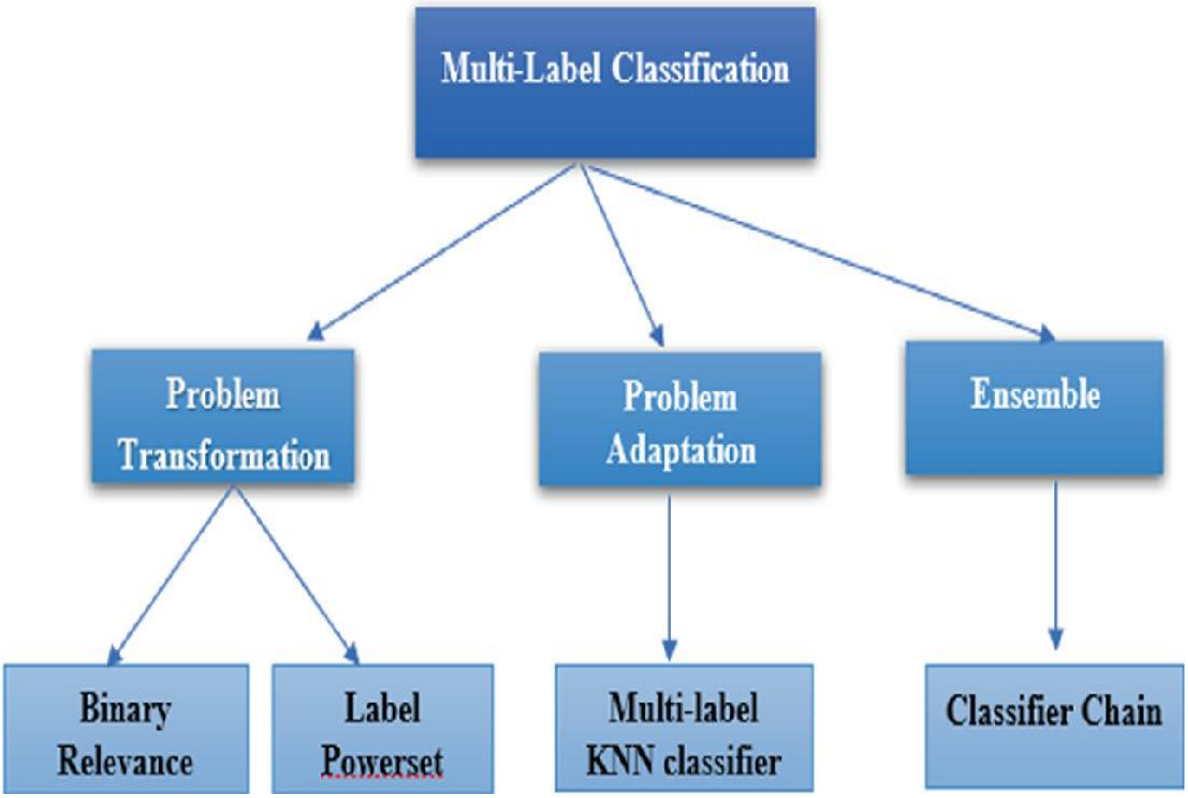


Fig: Multilabel classification

We will be studying each type in detail now.

- ## Problem Transformation

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.

X	Y_1	Y_2	Y_3	Y_4
$x^{(1)}$	0	1	1	0
$x^{(2)}$	1	0	0	0
$x^{(3)}$	0	1	0	0
$x^{(4)}$	1	0	0	1
$x^{(5)}$	0	0	0	1

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

X	Y_1
$x^{(1)}$	0
$x^{(2)}$	1
$x^{(3)}$	0
$x^{(4)}$	1
$x^{(5)}$	0

X	Y_2
$x^{(1)}$	1
$x^{(2)}$	0
$x^{(3)}$	1
$x^{(4)}$	0
$x^{(5)}$	0

X	Y_3
$x^{(1)}$	1
$x^{(2)}$	0
$x^{(3)}$	0
$x^{(4)}$	0
$x^{(5)}$	0

X	Y_4
$x^{(1)}$	0
$x^{(2)}$	0
$x^{(3)}$	0
$x^{(4)}$	1
$x^{(5)}$	1

Strong sides of binary relevance: - estimates single-label classifiers - can generalize beyond available label combinations

Weak sides of binary relevance: - not suitable for large number of labels - ignores label relations

2) Label Powerset

In this, we transform the problem into a multi-class problem with one multi-class classifier is trained on all unique label combinations found in the training data.

Let's understand it by an example.

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0
x4	0	1	1	0
x5	1	1	1	1
x6	0	1	0	0

In this, we find that x1 and x4 have the same labels; similarly, x3 and x6 have the same set of labels. So, label power set transforms this problem into a single multi-class problem as shown below.

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

X	y1
x1	1
x2	2
x3	3
x4	1
x5	4
x6	3

Thus label powerset transforms multi-label problem to a multi-class problem where each label combination is a separate class and uses a multi-class classifier to solve the problem.

Strong sides: - estimates label dependencies, with only one classifier - often best solution for subset accuracy if training data contains all relevant label combinations

Weak sides: - requires all label combinations predictable by the classifier to be present in the training data - very prone to under fitting with large label spaces

- ## Problem adaptation

1) Multi label Knn classifier

Adapted algorithm, as the name suggests, adapting the algorithm to directly perform multi-label classification, rather than transforming the problem into different subsets of problems.

MLkNN builds uses k-NearestNeighbors find nearest examples to a test class and uses Bayesian inference to select assigned labels. The multi-label version of kNN is represented by MLkNN

For each instance in the test set, its K nearest neighbours in the train set are identified. For each class y in Y the numbers of neighbouring instances belonging to y are used to compute the posterior probabilities that the test instance belongs or does not belong to y. Based on which of these probabilities are greater , we decide to assign or not the class y to a test instance.

For better understanding, assume that samples below are neighbours found out in the training set for test instance X = 1.

X	Y_1	Y_2
1	f_1	f_2
2	f_1	f_3
3	f_2	f_4
1	?	?

Now we have to prior and posteriori probabilities for each class for test instance X = 1

$$P(f_1|X = 1) = P(f_1) \cdot P(X=1|f_1) = 2/3 \cdot ½ = 1/3$$
$$P(f_2|X = 1) = P(f_2) \cdot P(X=1|f_2) = 2/3 \cdot ½ = 1/3$$
$$P(f_3|X = 1) = P(f_3) \cdot P(X=1|f_3) = 1/3 \cdot 0 = 0$$
$$P(f_4|X = 1) = P(f_4) \cdot P(X=1|f_4) = 1/3 \cdot 0 = 0$$

Maximal values were obtained for f1 and f2. Hence, we assign test instance to those two classes.

Strong sides: - estimates one multi-class sub classifier - works when distance between samples is a good predictor for label assignment - often used in biosciences.

Weak sides: - requires parameter estimation

- **Ensemble approach**

1) **Classifiers chain**

Ensemble always produces better results. Scikit-Multilearn library provides different ensembling classification functions, which you can use for obtaining better results.

In this, the first classifier is trained just on the input data and then each next classifier is trained on the input space and all the previous classifiers in the chain.

Let's try to this understand this by an example. In the dataset given below, we have X as the input space and Y's as the labels.

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

In classifier chains, this problem would be transformed into 4 different single label problems, just like shown below. Here yellow colored is the input space and the white part represents the target variable.

X	y1
x1	0
x2	1
x3	0

Classifier 1

X	y1	y2
x1	0	1
x2	1	0
x3	0	1

Classifier 2

X	y1	y2	y3
x1	0	1	1
x2	1	0	0
x3	0	1	0

Classifier 3

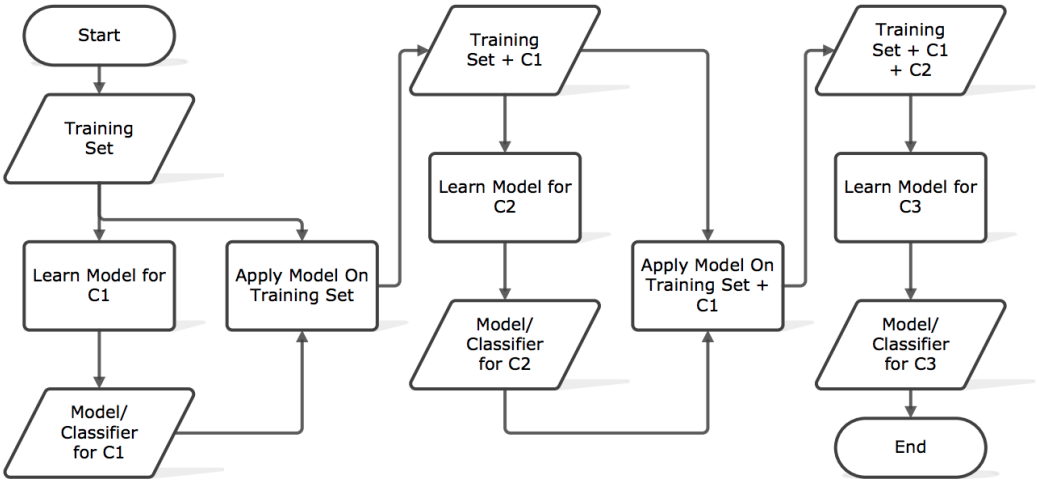
X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

Classifier 4

This is quite similar to binary relevance, the only difference being it forms chains in order to preserve label correlation.

The total number of classifiers needed for this approach is equal to the number of classes, but the training of the classifiers is more involved.

Following is an illustrated example with a classification problem of three categories {C1, C2, C3} chained in that order.



Strong sides: - estimates single-label classifiers - can generalize beyond available label combinations - takes label relations into account

Weak sides: - not suitable for large number of labels - quality strongly depends on the label ordering in chain.

- **Run and Evaluate selected models**

1) Binary relevance with GaussianNB as base classifier

Implementing the algorithm

```
# using binary relevance
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.naive_bayes import GaussianNB

# initialize binary relevance multi-label classifier
# with a gaussian naive bayes base classifier
classifier1 = BinaryRelevance(GaussianNB())

# train
classifier1.fit(x_train, y_train)
# predict
predictions = classifier1.predict(x_test)

# accuracy
acc_br=accuracy_score(y_test,predictions)*100
print("Accuracy = ",acc_br)
#cross validation score
cv_br=cross_val_score(classifier1,x,y,cv=3).mean()*100
print("Cross validation score=",cv_br)
print("\n")
```

Hamming loss and log loss for binary relevance

```
#Log loss
loss_br=log_loss(y_test,predictions.toarray())
print("log loss=",loss_br)
#hamming loss
hamm_loss_br=hamming_loss(y_test,predictions)
print("hamming loss=",hamm_loss_br)
print("\n")
```

F1 score, precision and auc-roc score

```
#f1 score
f1_br=f1_score(y_test,predictions.toarray(),average='micro')
print("f1 score=",f1_br)
#precision score micro averaged
pre_br=precision_score(y_test,predictions.toarray(),average='micro')
print("average precision score=",pre_br)
#auc roc curve
auc_br=roc_auc_score(y_test,predictions.toarray())
print("AUC ROC score=",auc_br)
print("\n")
```

Label ranking and coverage error

```
#label ranking
lr_br=label_ranking_average_precision_score(y_test,predictions.toarray())
print(" label ranking average precision score=",lr_br)
#coverage error
ce_br=coverage_error(y_test,predictions.toarray())
print("coverage area=",ce_br)
print("\n")
```

Evaluation metrics for Binary relevance classifier

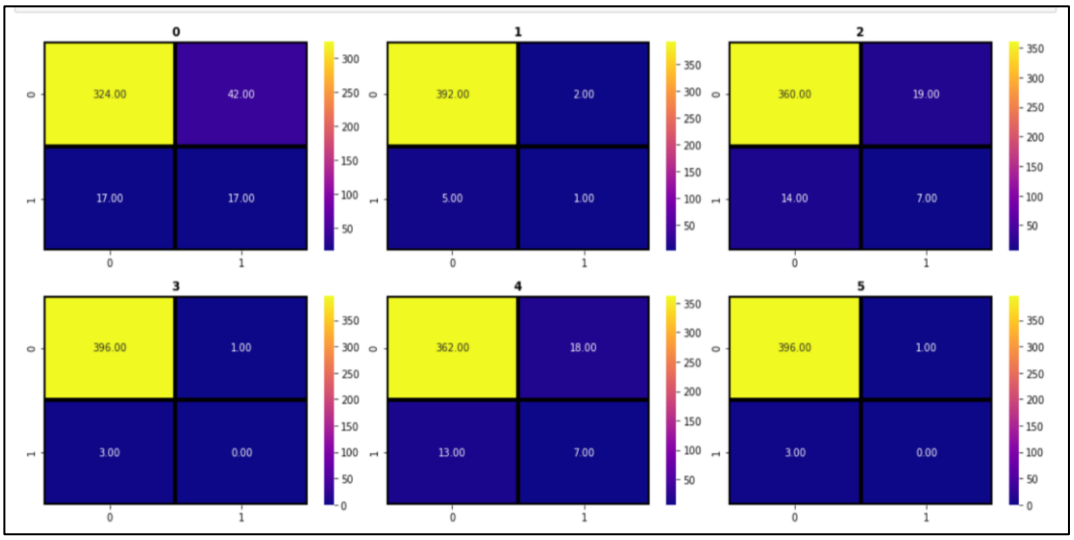
```
Accuracy =  80.75
Cross validation score= 83.04931618274946


log loss= 2.133116470314571
hamming loss= 0.0575


f1 score= 0.3168316831683169
average precision score= 0.2782608695652174
AUC ROC score= 0.5939693024131705


 label ranking average precision score= 0.9576250000000003
coverage area= 0.515
```

Confusion matrix for Binary relevance classifier

```
#confusion matrix for binary relevance
conf_mat1=multilabel_confusion_matrix(y_test,predictions)
conf_mat1
```

```
array([[[324,  42],
        [ 17,  17]],

       [[392,   2],
        [  5,   1]],

       [[360,  19],
        [ 14,   7]],

       [[396,   1],
        [  3,   0]],

       [[362,  18],
        [ 13,   7]],

       [[396,   1],
        [  3,   0]]], dtype=int64)
```

Heatmap of the confusion matrix:



2) Label power set

Implementing the algorithm

```
#create and fit classifier
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB

# initialize Label Powerset multi-label classifier
# with a gaussian naive bayes base classifier
classifier3 = LabelPowerset(GaussianNB())
classifier3.fit(x_train, y_train)
#predictions
predictions = classifier3.predict(x_test)
```

Hamming loss and log loss for label power set

```
#Log loss
loss_lps=log_loss(y_test,predictions.toarray())
print("log loss=",loss_lps)
#hamming loss
hamm_loss_lps=hamming_loss(y_test,predictions)
print("hamming loss=",hamm_loss_lps)
print("\n")
```

F1 score, precision and auc-roc score

```
#auc roc score
auc_lps=roc_auc_score(y_test,predictions.toarray())
print("AUC ROC score=",auc_lps)
#f1 score
f1_lps=f1_score(y_test,predictions.toarray(),average='micro')
print("f1 score=",f1_lps)
#precision score micro averaged
pre_lps=precision_score(y_test,predictions.toarray(),average='micro')
print("average precision score=",pre_lps)
print("\n")
```

Label ranking and coverage error

```
#coverage error
ce_lps=coverage_error(y_test,predictions.toarray())
print("coverage area=",ce_lps)
#label ranking
lr_lps=label_ranking_average_precision_score(y_test,predictions.toarray())
print(" label ranking average precision score=",lr_lps)
```

Evaluation metrics for label powerset

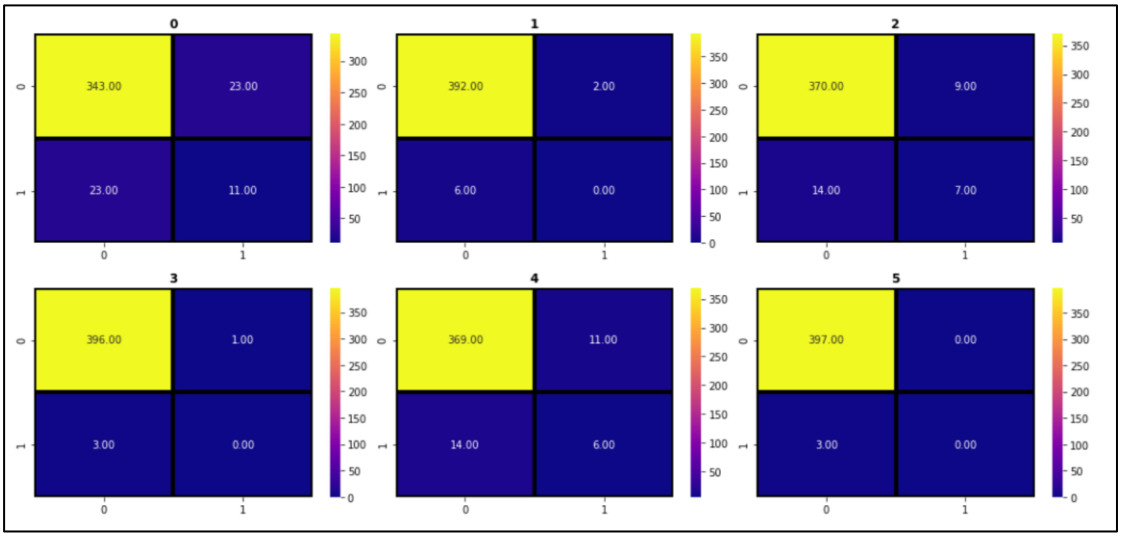
```
Accuracy =  85.5
Cross validation score= 86.45031838435138


log loss= 1.7382644951164647
hamming loss= 0.04541666666666667


AUC ROC score= 0.5694776759129783
f1 score= 0.3057324840764331
average precision score= 0.34285714285714286


coverage area= 0.5275
 label ranking average precision score= 0.9532083333333337
```

Confusion matrix for Label power set classifier

```
#confusion matrix for label powerset
conf_mat2=multilabel_confusion_matrix(y_test,predictions)
conf_mat2
```

```
array([[[343,  23],
        [ 23,  11]],

       [[392,   2],
        [  6,   0]],

       [[370,   9],
        [ 14,   7]],

       [[396,   1],
        [  3,   0]],

       [[369,  11],
        [ 14,   6]],

       [[397,   0],
        [  3,   0]]], dtype=int64)
```

Heatmap of the confusion matrix



3) Classifier chain

Implementing the algorithm

```
#create and fit classifier
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.problem_transform import ClassifierChain
classifier2 = ClassifierChain(MultinomialNB())
classifier2.fit(x_train, y_train)

#predictions
predictions = classifier2.predict(x_test)
# accuracy
acc_cc=accuracy_score(y_test,predictions)*100
print("Accuracy = ",acc_cc)
#cross validation score
cv_cc=cross_val_score(classifier2,x,y,cv=3).mean()*100
print("Cross validation score=",cv_cc)
print("\n")
```

Hamming loss and log loss for classifier chain

```
#log loss
loss_cc=log_loss(y_test,predictions.toarray())
print("log loss=",loss_cc)
#hamming loss
hamm_loss_cc=hamming_loss(y_test,predictions)
print("hamming loss=",hamm_loss_cc)
print("\n")
```

F1 score, precision and auc-roc score

```
#auc roc score
auc_cc=roc_auc_score(y_test,predictions.toarray())
print("AUC ROC score=",auc_cc)
#f1 score
f1_cc=f1_score(y_test,predictions.toarray(),average='micro')
print("f1 score=",f1_cc)
#precision score micro averaged
pre_cc=precision_score(y_test,predictions.toarray(),average='micro')
print("average precision score=",pre_cc)
print("\n")
```

Label ranking and coverage error

```
#coverage error
ce_cc=coverage_error(y_test,predictions.toarray())
print("coverage area=",ce_cc)

#label ranking
lr_cc=label_ranking_average_precision_score(y_test,predictions.toarray())
print(" label ranking average precision score=",lr_cc)
```

Evaluation metrics for classifier chain

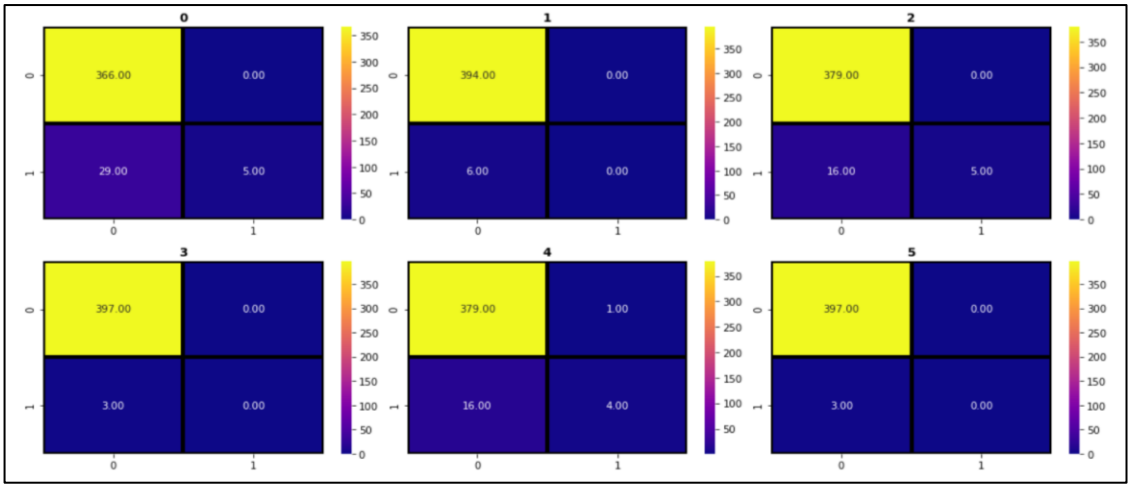
```
Accuracy =  91.25
Cross validation score= 90.65024544784664


log loss= 0.6192897523451344
hamming loss= 0.030833333333333334


AUC ROC score= 0.5485435402231068
f1 score= 0.27450980392156865
average precision score= 0.9333333333333333


coverage area= 0.525
 label ranking average precision score= 0.9502083333333334
```

Confusion matrix for classifier chain

```
#confusion matrix for classifier chain
conf_mat3=multilabel_confusion_matrix(y_test,predictions)
conf_mat3
```

```
array([[[366,   0],
        [ 29,   5]],

       [[394,   0],
        [  6,   0]],

       [[379,   0],
        [ 16,   5]],

       [[397,   0],
        [  3,   0]],

       [[379,   1],
        [ 16,   4]],

       [[397,   0],
        [  3,   0]]], dtype=int64)
```

Heatmap of the confusion matrix



4) Adapted Algorithm

Implementing the algorithm

```
from skmultilearn.adapt import MLkNN
classifier4 = MLkNN()
# train
classifier4.fit(x_train, y_train.values)
#predictions
predictions = classifier4.predict(x_test)

# accuracy
acc_mlknn=accuracy_score(y_test,predictions)*100
print("Accuracy = ",acc_mlknn)
#cross validation score
cv_mlknn=cross_val_score(classifier4,x,y.values,cv=3).mean()*100
print("Cross validation score=",cv_mlknn)
print("\n")
```

Hamming loss and log loss for adapted algorithm

```
#log loss
loss_mlknn=log_loss(y_test,predictions.toarray())
print("log loss=",loss_mlknn)
#hamming loss
hamm_loss_mlknn=hamming_loss(y_test,predictions)
print("hamming loss=",hamm_loss_mlknn)
print("\n")
```

F1 score, precision and auc-roc score

```
#auc roc score
auc_mlknn=roc_auc_score(y_test,predictions.toarray())
print("AUC ROC score=",auc_mlknn)
#f1 score
f1_mlknn=f1_score(y_test,predictions.toarray(),average='micro')
print("f1 score=",f1_mlknn)
#precision score micro averaged
pre_mlknn=precision_score(y_test,predictions.toarray(),average='micro')
print("average precision score=",pre_mlknn)
print("\n")
```

Label ranking and coverage error

```
#label ranking
lr_mlknn=label_ranking_average_precision_score(y_test,predictions.toarray())
print(" label ranking average precision score=",lr_mlknn)
#coverage error
ce_mlknn=coverage_error(y_test,predictions.toarray())
print("coverage area=",ce_mlknn)
```

Evaluation metrics for adapted algorithm

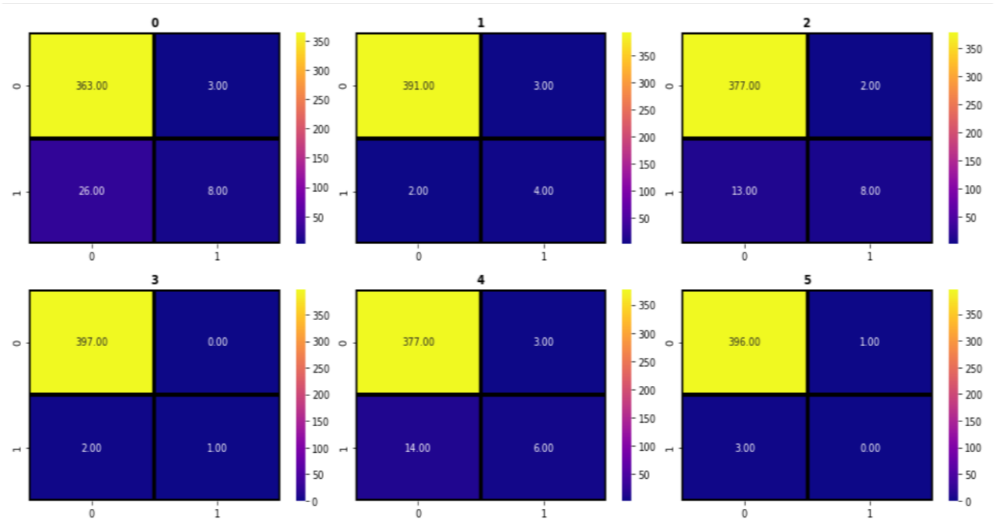
```
Accuracy =  91.0
Cross validation score= 88.6005945975961


log loss= 0.5322696133658981
hamming loss= 0.03


AUC ROC score= 0.6570620742254459
f1 score= 0.4285714285714286
average precision score= 0.6923076923076923


 label ranking average precision score= 0.9508229166666666
coverage area= 0.515
```

Confusion matrix for adapted algorithm

```
#confusion matrix for mlknn
conf_mat4=multilabel_confusion_matrix(y_test,predictions)
conf_mat4
```

```
array([[[363,   3],
        [ 26,   8]],

       [[391,   3],
        [  2,   4]],

       [[377,   2],
        [ 13,   8]],

       [[397,   0],
        [  2,   1]],

       [[377,   3],
        [ 14,   6]],

       [[396,   1],
        [  3,   0]]], dtype=int64)
```

Heatmap of the confusion matrix



- # Key Metrics for success in solving problem under consideration

Choosing the right metric is crucial while evaluating machine learning (ML) models. Various metrics are proposed to evaluate ML models in different applications .In some applications looking at a single metric may not give you the whole picture of the problem you are solving, and you may want to use a subset of the metrics.

There are various ways to evaluate a classification model, and we will be covering some of the most popular ones below.

1) Classification Accuracy

Classification accuracy is perhaps the simplest metrics one can imagine, and is defined as the number of correct predictions divided by the total number of predictions, multiplied by 100. So in the above example, out of 1100 samples 1030 are predicted correctly, resulting in a classification accuracy of:

Classification accuracy= (90+940)/ (1000+100) = 1030/1100= 93.6%

2) Confusion Matrix (not a metric, but important to know!)

One of the key concept in classification performance is confusion matrix (AKA error matrix), which is a tabular visualization of the model predictions versus the ground-truth labels. Each row of confusion matrix represents the instances in a predicted class and each column represents the instances in an actual class.

Confusion Matrix is a 2X2 Matrix where:

- **TRUE POSITIVE**: measures the proportion of actual positives that are correctly identified.
- **TRUE NEGATIVE**: measures the proportion of actual positives that are not correctly identified.
- **FALSE POSITIVE**: measures the proportion of actual negatives that are correctly identified.
- **FALSE NEGATIVE**: measures the proportion of actual negatives that are not correctly identified.

In Multilabel confusion matrix MCM, the count of true negatives is MCM (0, 0), false negatives is MCM :,(1, 0), true positives is MCM (1, 1) and false positives is MCM (0, 1).

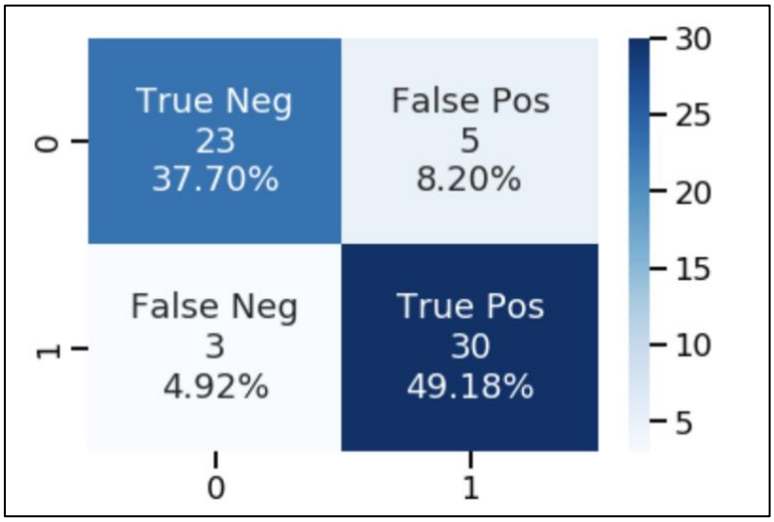


Fig: Example of confusion matrix for multi-label

3) Hamming loss

It is important to note that unlikely accuracy in hamming loss the smaller the result is the better is the model. For an example if hamming loss, is 0.32 which means that if you are trying to predict the aspects of 100 sentences the model will predict incorrectly about 32% of the independent aspects.
Hamming loss value ranges from 0 to 1. Lesser value of hamming loss indicates a better classifier. It is the fraction of labels that are incorrectly predicted and brings additional information in case of unbalanced datasets.

4) F1 score

F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall and there is an uneven class distribution (large number of Actual Negatives) as in our example.

F1-score= 2*Precision*Recall/ (Precision + Recall)

The best value is 1 and the worst value is 0.

5) Log loss

This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of a logistic model that returns y_pred probabilities for its training data y_true. The log loss is only defined for two or more labels. The bolder the probabilities, the better will be your Log Loss — closer to zero. It is a measure of uncertainty (you may call it entropy), so a low Log Loss means a low uncertainty/entropy of your model.

6) **Precision**

The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

Precision= True Positive/ (True Positive+ False Positive)

The best value is 1 and the worst value is 0.

7) **Coverage error**

Compute how far we need to go through the ranked scores to cover all true labels. The best value is equal to the average number of labels in y_true per sample.

The best value of coverage is when it is equal to average number of true class labels.

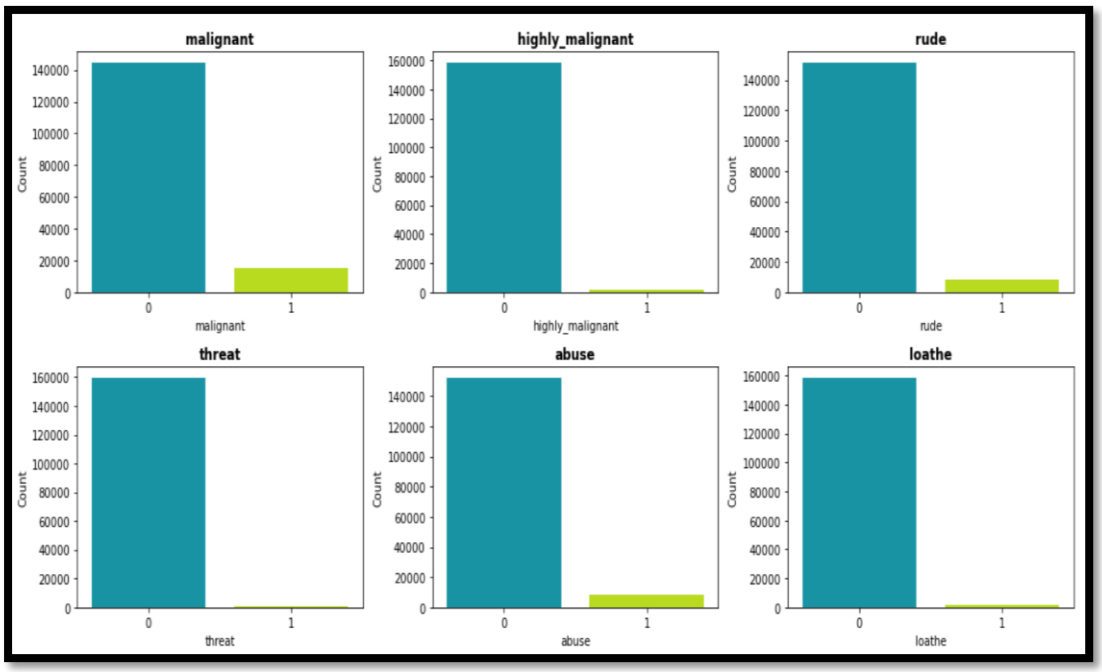
The best performance is achieved with a ranking loss of zero.

8) **Label ranking average precision (LRAP)**

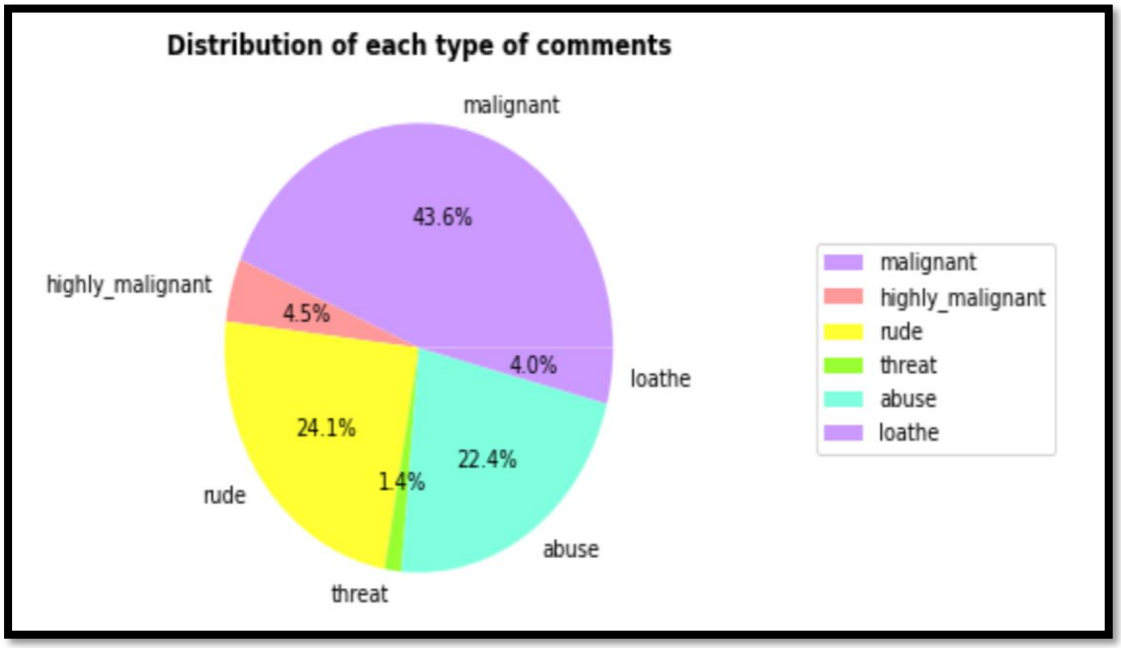
It is the average over each ground truth label assigned to each sample, of the ratio of true vs. total labels with lower score.
This metric is used in Multilabel ranking problem, where the goal is to give better rank to the labels associated to each sample.
The obtained score is always strictly greater than 0 and the best value is 1.

- **Visualizations**

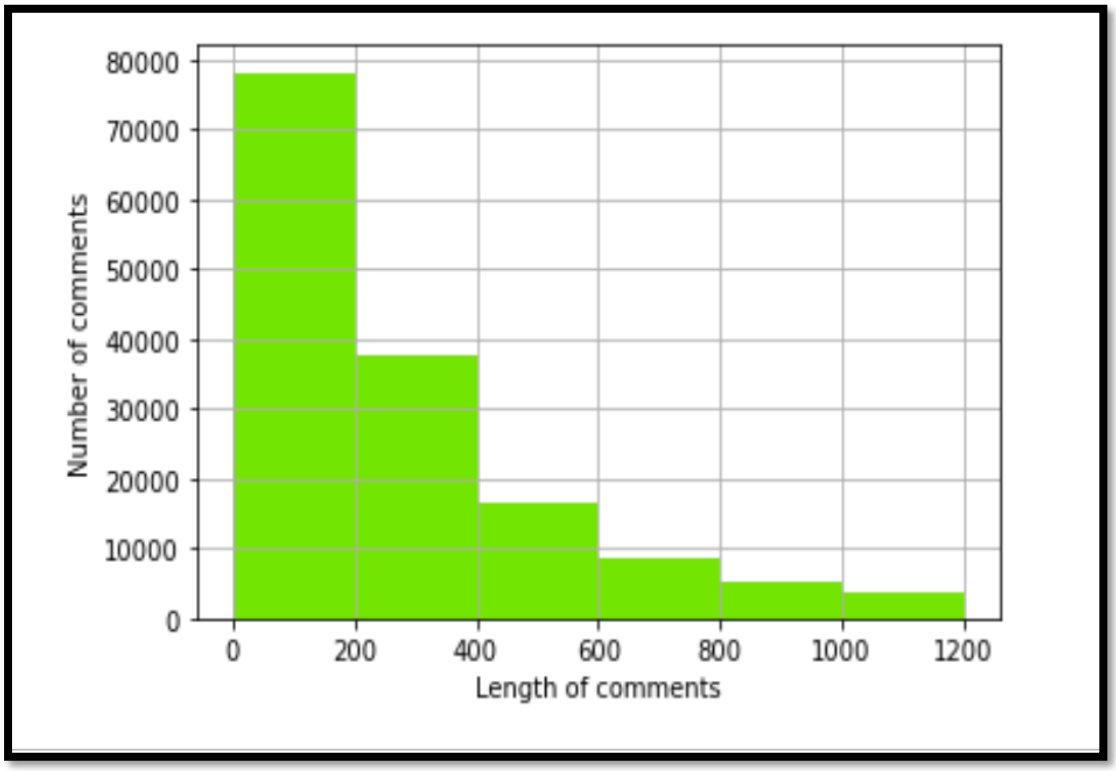
Plotting the value count of each type of comment:



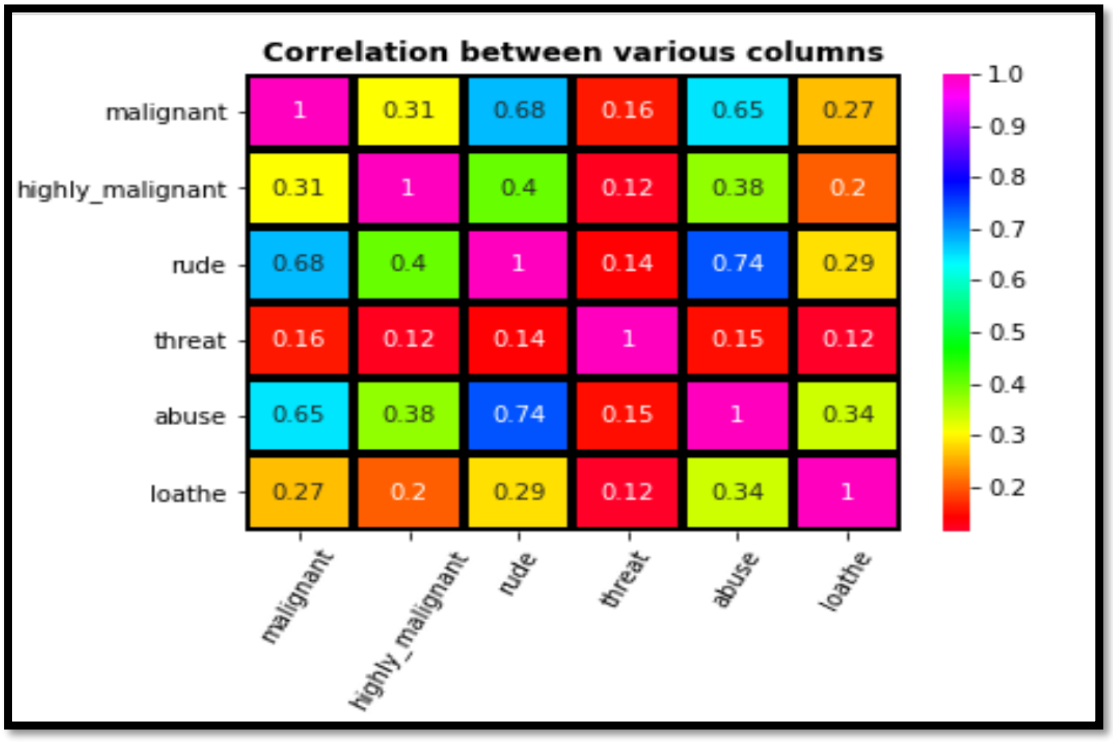
Distribution of comments:



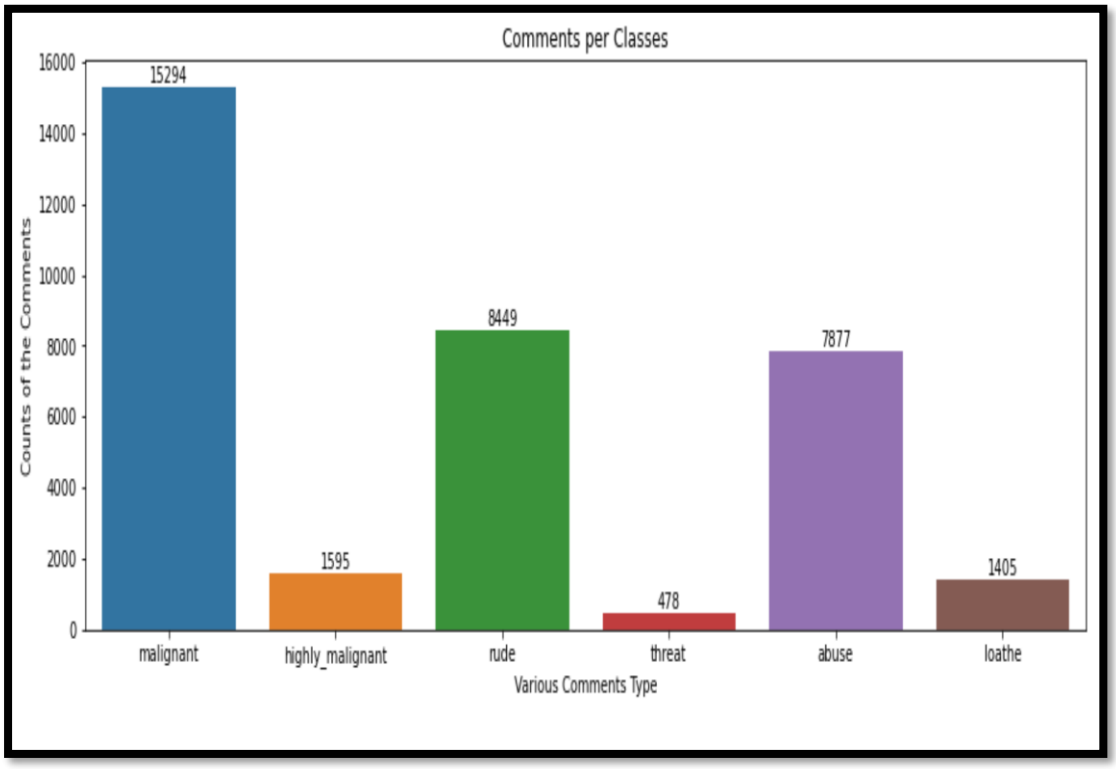
Average length of comments:



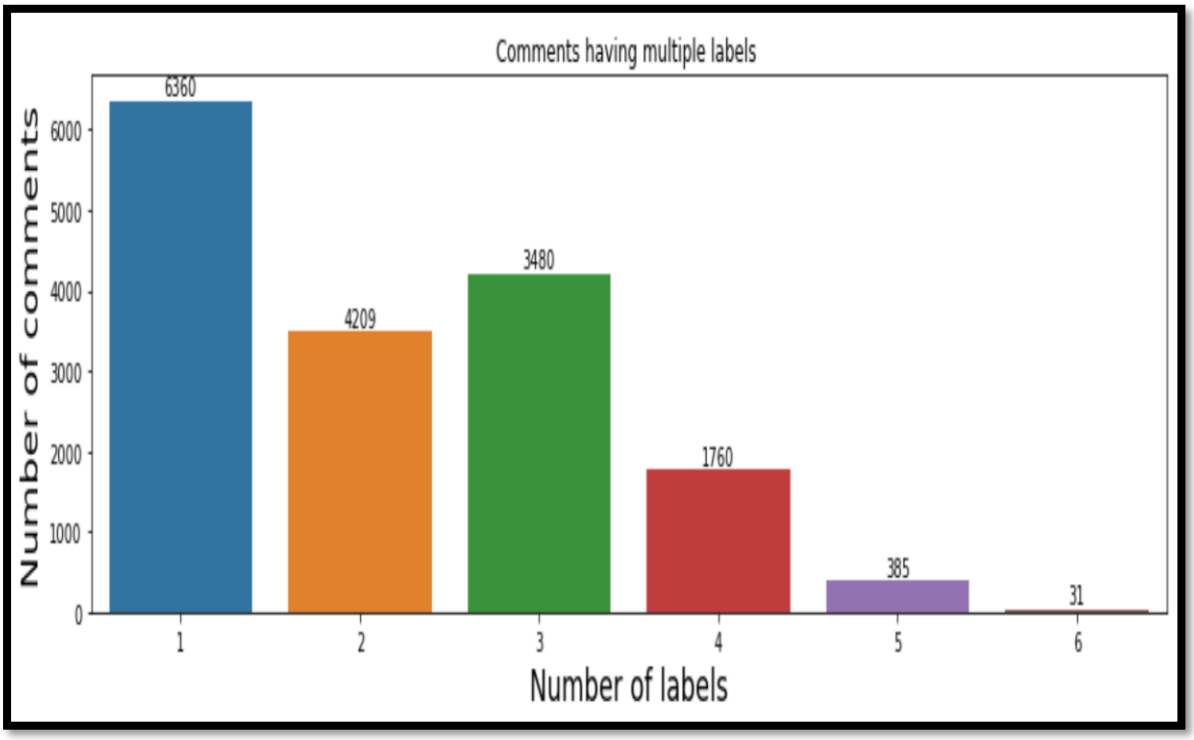
Correlation matrix



Then we count the number of comments under each label.

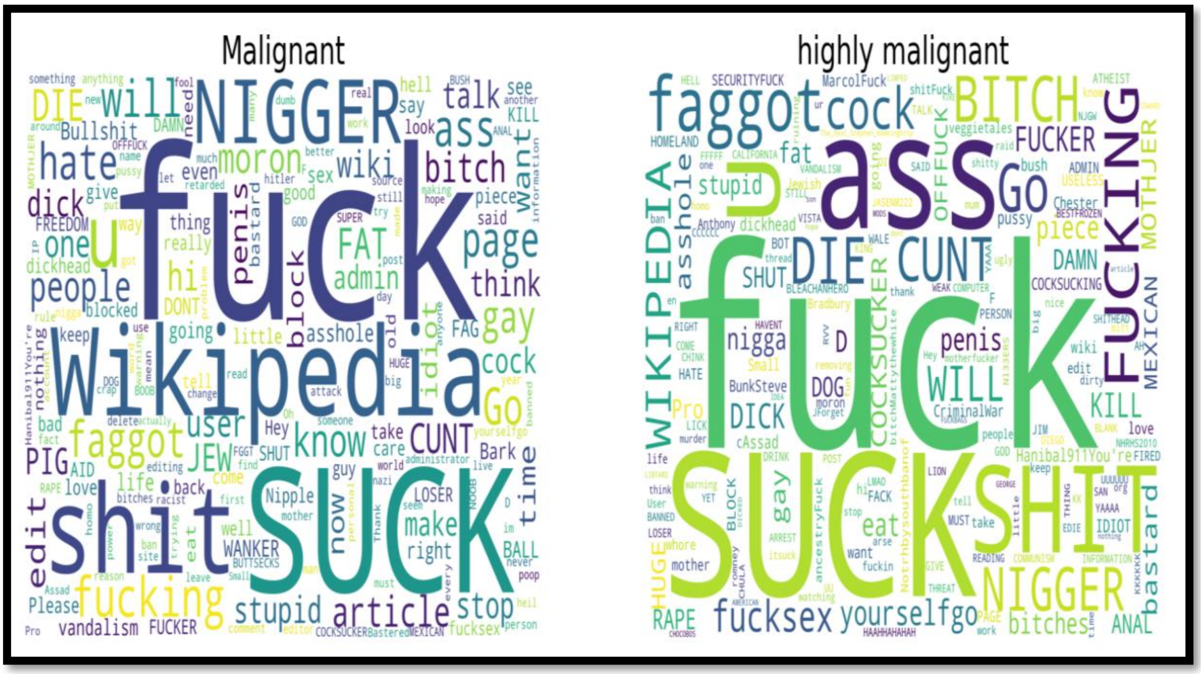


Counting the number of comments having multiple labels:

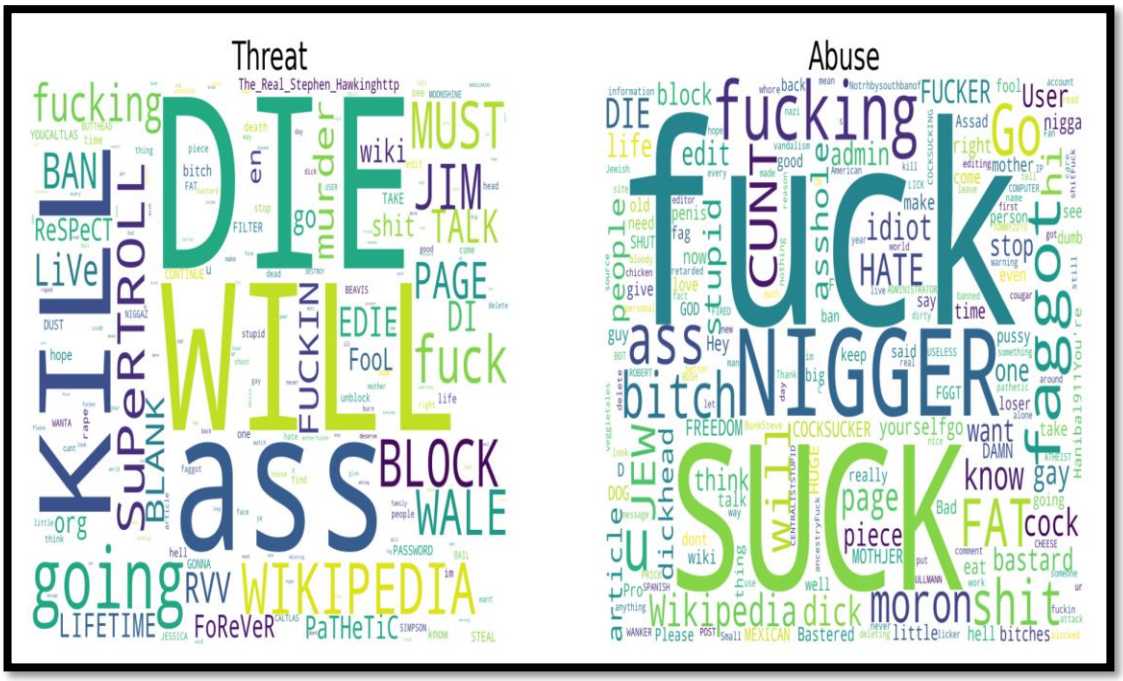


WordCloud representation of most used words in all comments.

WordCloud representation of most used words in each category of comments.



WordCloud representation of most used words in threat and abuse kind of comments.



WordCloud representation of most used words in rude kind of comments.

WordCloud representation of most used words in rude kind of comments.

- **Interpretation of the Results**

  After pre-processing the data and all data exploration, we implemented various algorithms for multi-label classification and got various inferences from the data.

- There were 6360 comments which fall into only 1 category of the comment type.4209 comments had 2 labels for example malignant and rude.3480 comments had 3 labels for example malignant, abuse and rude. There were 31 comments which had all 6 types of comments categories at the same time.
- There were 15294 malignant comments which are maximum. Rude comments were second highest. Threat comments count was only 478 which is the least. Highly malignant and loathe comments were moderate in number.
- Malignant comments had the highest share of all which was 43.6%.Rude comments were second highest and abuse had 22.4% of share in the total distribution. Loathe comments very least (4.0%)
- For the correlation; the highest co-relation was seen between rude--> abuse of 0.74.The second highest correlation is between malignant--->rude which is 0.68 malignant and abuse have a correlation of 0.65.Loathe and threat had the least correlations with all variables.
- We have observed all the evaluation metrics for each and every algorithm which can be concluded in the following table:

	Model	Accuracy Score	Cross Validation score	Hamming Loss	Log Loss	Auc-Roc score	F1 Score	Precision	coverage error	Label ranking
0	Classifier chain	91.25	90.650245	0.030833	0.619290	0.548544	0.274510	0.933333	0.5250	0.950208
1	Adapted algorithm	91.00	88.600595	0.030000	0.532270	0.657062	0.428571	0.692308	0.5150	0.950823
2	Label powerset	85.50	86.450318	0.045417	1.738264	0.569478	0.305732	0.342857	0.5275	0.953208
3	Binary Relevance	80.75	83.049316	0.057500	2.133116	0.593969	0.316832	0.278261	0.5150	0.957625

- The classifier chain algorithm is saved as our final model as it is significantly working well in all the evaluation metrics. Classifier chain gave us an accuracy of 91.25 and cross validation of 90.6.log loss= 0.61 and hamming loss= 0.03.AUC ROC score= 0.54 f1 score=0.27 and average precision score= 0.93 and a good label ranking average precision score of 0.95.
- The confusion matrix for classifier chain was observed and following were the results:

  For label (1) i.e. malignant class there are 5 True positives and 366 True Negatives and 29 error values (FP and FN).

  For label (2) i.e. highly malignant class there is 0 True positives and 394 True Negatives and 6 error values (FP and FN)

  For label (3) i.e. rude class there are 5 True positives and 379 True Negatives and 16 error values (FP and FN)

  For label (4) i.e. threat class there is 0 True positives and 397 True Negatives and 3 erroneous value (FP and FN)

  For label (5) i.e. abuse class there are 4 True positives and 379 True Negatives and (16+1) i.e. 17 erroneous values (FP and FN)

  For label (6) i.e. loathe class there is 0 True positives and 397 True Negatives and 3 erroneous values (FP and FN)

- The predicted values yield us a sparse matrix as multiple number of elements were zero hence we have converted it into an array before saving it to data frame.

- After classifier chain the adaptive algorithm can be seen performing well with an accuracy of 91.0 and cross validation of 88.log loss= 0.5321 and hamming loss= 0.03.AUC ROC score= 0.659.f1 score=0.42 and average precision score= 0.69 and a good label ranking average precision score of 0.95.

- We can see very minute difference between the classifier chain and adaptive algorithm but we have finalized classifier chain to avoid problem of over-fitting.

- As compared to classifier chain there is more difference between accuracy and CV for adaptive classifier. Hence, the decision to choose classifier chain.

CONCLUSION

The aim of this project is to create a tool that can help in the fight against online harassment. We will create a model that can not only decide whether a comment is toxic, but also describe the kind of toxicity that is present. This would allow online platforms to selectively remove the particular kind of toxicity they do not like. For example, an online community could be fine with mild swearing, but be completely opposed to obscenities.

As cyber bullying is a major societal problem. Malignant comment classifier helps to filter out toxic words to make online world a safer and more harmonious place. The research done in this paper is intended to enhance fair online talk and views sharing in social media.

There are two main methods for tackling a multi-label classification problem: problem transformation methods and algorithm adaptation methods.

Problem transformation methods transform the multi-label problem into a set of binary classification problems, which can then be handled using single-class classifiers.

Whereas algorithm adaptation methods adapt the algorithms to directly perform multi-label classification. In other words, rather than trying to convert the problem to a simpler problem, they try to address the problem in its full form.

Out of Memory, Kernel Died! Was the most common error i encountered throughout the project. It would follow me whichever direction i moved. On applying even the most basic algorithm on the most basic method, the program would run for good ten minutes and finally end up with out of memory error. After scratching my head for two days, the idea finally triggered. Trim the data!!!!! Hence we have used only 2000 random records to build and study the model

The classifier chain algorithm is saved as our final model as it is significantly working well in all the evaluation metrics. Classifier chain gave us an accuracy of 91.25 and cross validation of 90.6.log loss= 0.61 and hamming loss= 0.03.AUC ROC score= 0.54 f1 score=0.27 and average precision score= 0.93 and a good label ranking average precision score of 0.95.

Our project work has shown that harmful or toxic comments in the social media space have many negative impacts to society. The ability to readily and accurately identify comments as toxic could provide many benefits while mitigating the harm. Also, our project has shown the capability of readily available algorithms to be employed in such a way to address this challenge. In our specific study, it was demonstrated that claasifierchain provides substantial improvement in classification versus another algorithms. To sum up, popular methods for Multilabel classification were described and compared on real-life example. The best results were obtained for Classifier Chain method

- ## Limitations of this work and Scope for Future Work

One of the main undiscovered issues is how to identify algorithms that are able to implement high sensitivity in detection of malignant comments. Of course, identifying comment that is not toxic as a toxic can be frustrating for the users and there should be a lot of effort to form an algorithm with highest degree of sensitivities.

Sarcasm and irony kind of comments detection is a hard task it increases difficulty of comment classification, because the texts usually state the opposite of what is really meant.

In problem transformation classifiers we often need to estimate not only a hyper parameter, but also the parameter of the base classifier, and also - maybe even the problem transformation method, which makes it a tedious job.

Future scope

The current project will only predict the type or toxicity in the comments. In future we can add the following features:

- Analyse which age group is being toxic towards a particular group or brand.
- Add feature to automatically sensitize words which are classified as toxic.
- Automatically send alerts to the concerned authority if comments are classified as highly malignant.
- Build a feedback loop to further increase the efficiency of the model.
- Handle mistakes and short forms of words to get better accuracy of the result.
- The same problem can be solved using LSTMs in deep learning.
- For more speed we could use decision trees and for a reasonable trade-off between speed and accuracy we could also opt for ensemble models.
- Other frameworks such as MEKA can be used to deal with multi-label classification problems.
- In addition, new regulations in certain European countries have been already established enforcing to delete illegal content in less than 72 hours which can be implemented everywhere