Section 1:

1. Bag of Words Model.

```
from keras.preprocessing.text import Tokenizer

text = [
  'There was a man',
  'The man had a dog',
  'The dog and the man walked',
]
# using tokenizer
model = Tokenizer()
model.fit_on_texts(text)

#print keys
print(f'Key : {list(model.word_index.keys())}')

#create bag of words representation
rep = model.texts_to_matrix(text, mode='count')
print(rep)
```

Output:

```
Key : ['man', 'the', 'a', 'dog', 'there', 'was', 'had', 'and',
'walked']
[[0. 1. 0. 1. 0. 1. 1. 0. 0. 0.]
 [0. 1. 1. 1. 1. 0. 0. 1. 0. 0.]
 [0. 1. 2. 0. 1. 0. 0. 0. 1. 1.]]
```

2. Create a Bag of Words Model with Sklearn

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"



CountVec = CountVectorizer(ngram_range=(1,1), # to use bigrams ngram_range=(2,2)
                           stop_words='english')
#transform
Count_data = CountVec.fit_transform([sentence_1,sentence_2])
#print(CountVec)
#print(Count_data)

#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
print(cv_dataframe)
```

```
   good  job  miss
0     1    1     1
1     1    0     0
```

3. Feature Extraction with Tf-Idf vectorizer

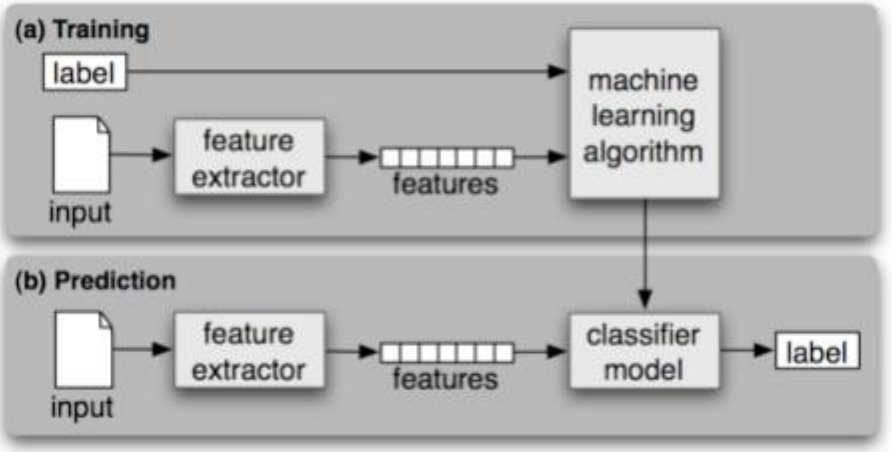
We can use the TfidfVectorizer() function from the Sk-learn library to easily implement the above BoW(Tf-IDF), model.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"

#without smooth IDF
print("Without Smoothing:")
#define tf-idf
tf_idf_vec = TfidfVectorizer(use_idf=True,
                        smooth_idf=False,
                        ngram_range=(1,1),stop_words='english') # to use only  bigrams ngram_range=(2,2)
#transform
tf_idf_data = tf_idf_vec.fit_transform([sentence_1,sentence_2])

#create dataframe
tf_idf_dataframe=pd.DataFrame(tf_idf_data.toarray(),columns=tf_idf_vec.get_feature_names())
print(tf_idf_dataframe)
print("\n")

#with smooth
tf_idf_vec_smooth = TfidfVectorizer(use_idf=True,
                        smooth_idf=True,
                        ngram_range=(1,1),stop_words='english')

tf_idf_data_smooth = tf_idf_vec_smooth.fit_transform([sentence_1,sentence_2])

print("With Smoothing:")
tf_idf_dataframe_smooth=pd.DataFrame(tf_idf_data_smooth.toarray(),columns=tf_idf_vec_smooth.get_feature_names())
print(tf_idf_dataframe_smooth)
```

Output:

```
Without Smoothing:
       good       job      miss
0  0.385372  0.652491  0.652491
1  1.000000  0.000000  0.000000


With Smoothing:
       good       job      miss
0  0.449436  0.631667  0.631667
1  1.000000  0.000000  0.000000
```

4. Supervised Classification



1. EDA

In [1]:
```
import os
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))
#printmd('**bold**')
```

In [3]:
```
data_path = "/Users/kartik/Desktop/AAIC/Projects/jigsaw-toxic-comment-classification-challenge/data/train.csv"
```

In [4]:
```
data_raw = pd.read_csv(data_path)
#data_raw = data_raw.loc[np.random.choice(data_raw.index, size=2000)]
data_raw.shape
```

Out[4]: (159571, 8)

In [5]:
```
print("Number of rows in data =",data_raw.shape[0])
print("Number of columns in data =",data_raw.shape[1])
print("\n")
printmd("**Sample data:**")
data_raw.head()
```

```
Number of rows in data = 159571
Number of columns in data = 8
```

Sample data:

Out[5]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
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

1.1. Checking for missing values

In [6]:
```
missing_values_check = data_raw.isnull().sum()
print(missing_values_check)
```

```
id               0
comment_text     0
toxic            0
severe_toxic     0
obscene          0
threat           0
insult           0
identity_hate    0
dtype: int64
```

1.2. Calculating number of comments under each label

In [7]:
```
# Comments with no label are considered to be clean comments.
# Creating seperate column in dataframe to identify clean comments.

# We use axis=1 to count row-wise and axis=0 to count column wise

rowSums = data_raw.iloc[:,2:].sum(axis=1)
clean_comments_count = (rowSums==0).sum(axis=0)

print("Total number of comments = ",len(data_raw))
print("Number of clean comments = ",clean_comments_count)
print("Number of comments with labels =",(len(data_raw)-clean_comments_count))
```

```
Total number of comments =  159571
Number of clean comments =  143346
Number of comments with labels = 16225
```

In [8]:
```
categories = list(data_raw.columns.values)
categories = categories[2:]
print(categories)
```

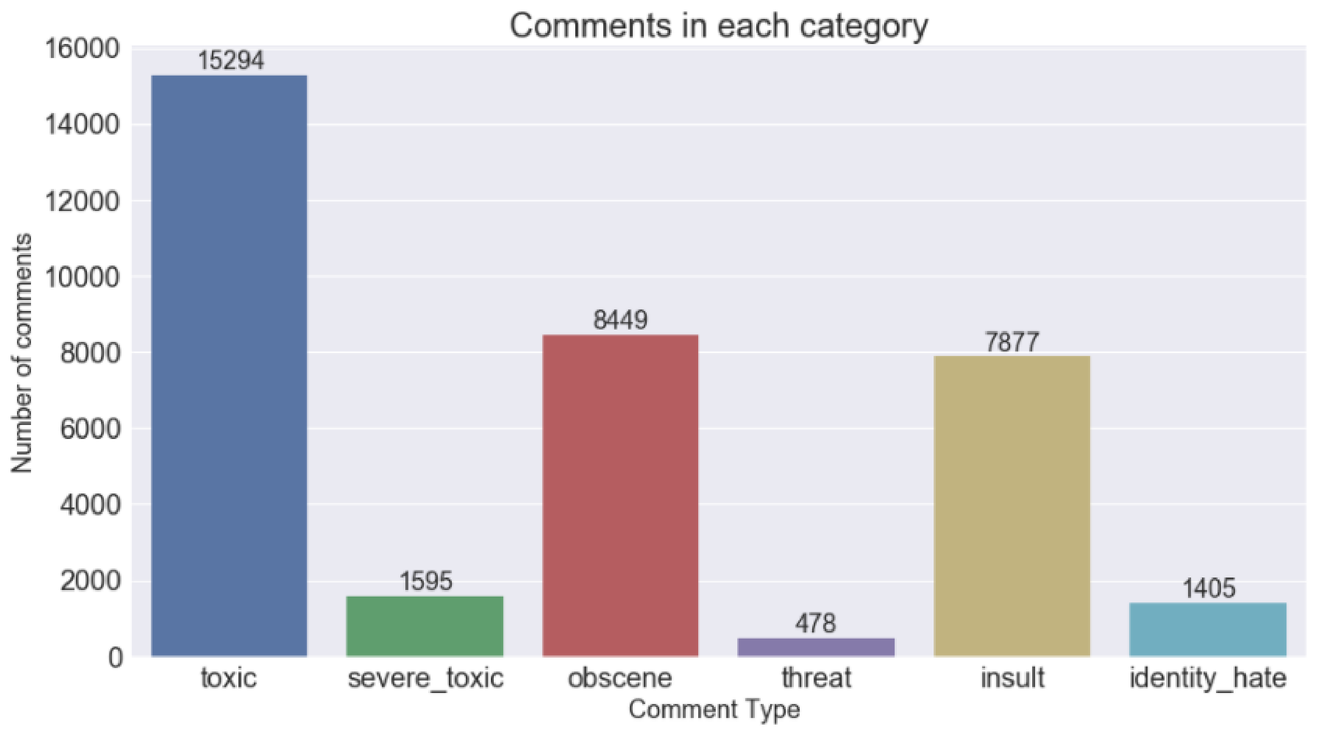
```
['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
```

In [9]:
```
# Calculating number of comments in each category

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

Out[9]:

	category	number of comments
0	toxic	15294
1	severe_toxic	1595
2	obscene	8449
3	threat	478
4	insult	7877
5	identity_hate	1405

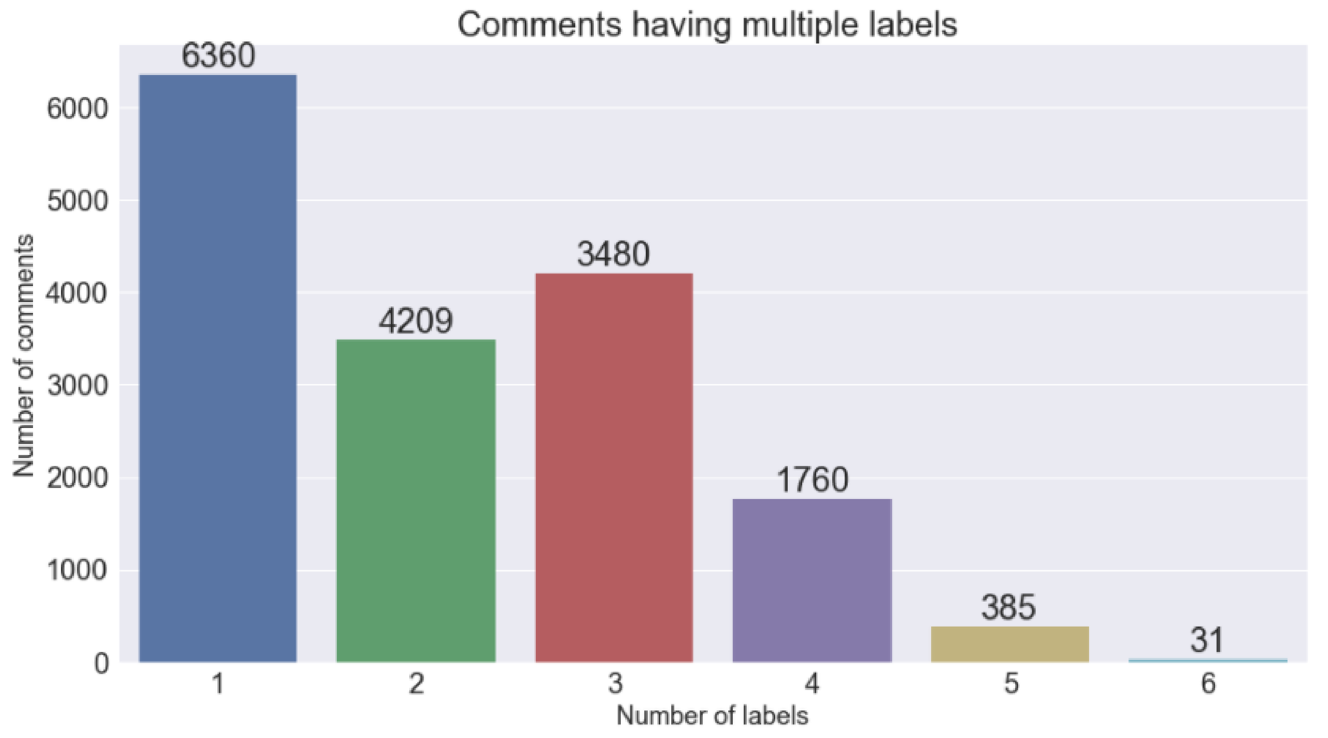
In [10]:
```
sns.set(font_scale = 2)
plt.figure(figsize=(15,8))

ax= sns.barplot(categories, data_raw.iloc[:,2:].sum().values)

plt.title("Comments in each category", fontsize=24)
plt.ylabel('Number of comments', fontsize=18)
plt.xlabel('Comment Type ', fontsize=18)

#adding the text labels
rects = ax.patches
labels = data_raw.iloc[:,2:].sum().values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom', fontsize=18)

plt.show()
```



1.3. Calculating number of comments having multiple labels

In [11]:

```
rowSums = data_raw.iloc[:,2:].sum(axis=1)
multiLabel_counts = rowSums.value_counts()
multiLabel_counts = multiLabel_counts.iloc[1:]

sns.set(font_scale = 2)
plt.figure(figsize=(15,8))

ax = sns.barplot(multiLabel_counts.index, multiLabel_counts.values)

plt.title("Comments having multiple labels ")
plt.ylabel('Number of comments', fontsize=18)
plt.xlabel('Number of labels', fontsize=18)

#adding the text labels
rects = ax.patches
labels = multiLabel_counts.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')

plt.show()
```



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

In [12]:
```
from wordcloud import WordCloud,STOPWORDS

plt.figure(figsize=(40,25))

# toxic
subset = data_raw[data_raw.toxic==1]
text = subset.comment_text.values
cloud_toxic = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))

plt.subplot(2, 3, 1)
plt.axis('off')
plt.title("Toxic",fontsize=40)
plt.imshow(cloud_toxic)


# severe_toxic
subset = data_raw[data_raw.severe_toxic==1]
text = subset.comment_text.values
cloud_severe_toxic = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))

plt.subplot(2, 3, 2)
plt.axis('off')
plt.title("Severe Toxic",fontsize=40)
plt.imshow(cloud_severe_toxic)
```

```python
# obscene
subset = data_raw[data_raw.obscene==1]
text = subset.comment_text.values
cloud_obscene = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))

plt.subplot(2, 3, 3)
plt.axis('off')
plt.title("Obscene",fontsize=40)
plt.imshow(cloud_obscene)


# threat
subset = data_raw[data_raw.threat==1]
text = subset.comment_text.values
cloud_threat = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))

plt.subplot(2, 3, 4)
plt.axis('off')
plt.title("Threat",fontsize=40)
plt.imshow(cloud_threat)
```

```python
# insult
subset = data_raw[data_raw.insult==1]
text = subset.comment_text.values
cloud_insult = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))

plt.subplot(2, 3, 5)
plt.axis('off')
plt.title("Insult",fontsize=40)
plt.imshow(cloud_insult)


# identity_hate
subset = data_raw[data_raw.identity_hate==1]
text = subset.comment_text.values
cloud_identity_hate = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))

plt.subplot(2, 3, 6)
plt.axis('off')
plt.title("Identity Hate",fontsize=40)
plt.imshow(cloud_identity_hate)

plt.show()
```

2. Data Pre-Processing

In [13]:
```
data = data_raw
data = data_raw.loc[np.random.choice(data_raw.index, size=2000)]
data.shape
```

Out[13]: (2000, 8)

In [14]:
```
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
import re

import sys
import warnings

if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

2.1. Cleaning Data

In [15]:
```
def cleanHtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(sentence))
    return cleantext


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


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
```

In [16]:
```
data['comment_text'] = data['comment_text'].str.lower()
data['comment_text'] = data['comment_text'].apply(cleanHtml)
data['comment_text'] = data['comment_text'].apply(cleanPunc)
data['comment_text'] = data['comment_text'].apply(keepAlpha)
data.head()
```

Out[16]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
108305	42fdb3027a4ca9f5	apart from adding the odd word here and there ...	0	0	0	0	0	0
65705	afbec12bbe7d9fda	no consensus take it to the npov noticeboard a...	0	0	0	0	0	0
137312	deaa002fef0e89c0	actually most of it is quite good there are a ...	0	0	0	0	0	0
48543	81d57be4398b9b1c	requested move dialect levelling dialect lev...	0	0	0	0	0	0
121368	896a0148135147b5	the rewrite is in used a lot of the same conte...	0	0	0	0	0	0

```
['zero','one','two','three','four','five','six','seven','eight','nine','ten','may','also','across','among','beside','however','yet','within']
```

2.2. Removing Stop Words

In [17]:
```
stop_words = set(stopwords.words('english'))
stop_words.update(['zero','one','two','three','four','five','six','seven','eight','nine','ten','may','also','across','among','beside','ho
wever','yet','within'])
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)

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

Out[17]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
108305	42fdb3027a4ca9f5	apart adding odd word hadnt touched ne...	0	0	0	0	0	0
65705	afbec12bbe7d9fda	consensus take npov noticeboard youll see...	0	0	0	0	0	0
137312	deaa002fef0e89c0	actually quite good errors need fixin...	0	0	0	0	0	0
48543	81d57be4398b9b1c	requested move dialect levelling dialect lev...	0	0	0	0	0	0
121368	896a0148135147b5	rewrite used lot content wikified res...	0	0	0	0	0	0

2.3. Stemming

In [18]:
```
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()
```

Out[18]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
108305	42fdb3027a4ca9f5	apart ad odd word hadnt touch near year	0	0	0	0	0	0
65705	afbec12bbe7d9fda	consensus take npov noticeboard youll see happen	0	0	0	0	0	0
137312	deaa002fef0e89c0	actual quit good error need fix includ critic ...	0	0	0	0	0	0
48543	81d57be4398b9b1c	request move dialect level dialect level brita...	0	0	0	0	0	0
121368	896a0148135147b5	rewrit use lot content wikifi restructur neutr...	0	0	0	0	0	0

2.4. Train-Test Split

In [19]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, random_state=42, test_size=0.30, shuffle=True)

print(train.shape)
print(test.shape)
```

```
(1400, 8)
(600, 8)
```

In [20]:
```
train_text = train['comment_text']
test_text = test['comment_text']
```

2.5. TF-IDF

In [21]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(strip_accents='unicode', analyzer='word', ngram_range=(1,3), norm='l2')
vectorizer.fit(train_text)
vectorizer.fit(test_text)
```

Out[21]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 3), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents='unicode', sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [22]:
```
x_train = vectorizer.transform(train_text)
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)
```

3. Multi-Label Classification

3.1. Multiple Binary Classifications - (One Vs Rest Classifier)

In [23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
```

In [24]:
```
%%time

# Using pipeline for applying logistic regression and one vs rest classifier
LogReg_pipeline = Pipeline([
                ('clf', OneVsRestClassifier(LogisticRegression(solver='sag'), n_jobs=-1)),
            ])

for category in categories:
    printmd('**Processing {} comments...**'.format(category))

    # Training logistic regression model on train data
    LogReg_pipeline.fit(x_train, train[category])

    # calculating test accuracy
    prediction = LogReg_pipeline.predict(x_test)
    print('Test accuracy is {}'.format(accuracy_score(test[category], prediction)))
    print("\n")
```

Processing toxic comments...

```
Test accuracy is 0.9
```

Processing severe_toxic comments...

```
Test accuracy is 0.9916666666666667
```

Processing obscene comments...

```
Test accuracy is 0.95
```

Processing threat comments...

```
Test accuracy is 0.995
```

Processing insult comments...

```
Test accuracy is 0.9516666666666667
```

Processing identity_hate comments...

```
Test accuracy is 0.9983333333333333
```

```
CPU times: user 512 ms, sys: 329 ms, total: 841 ms
Wall time: 1.37 s
```

3.2. Multiple Binary Classifications - (Binary Relevance)

In [25]:

```
%%time

# 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
classifier = BinaryRelevance(GaussianNB())

# train
classifier.fit(x_train, y_train)

# predict
predictions = classifier.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("\n")
```

```
Accuracy =  0.8566666666666667
```

```
CPU times: user 7.64 s, sys: 5.92 s, total: 13.6 s
Wall time: 13.6 s
```

3.3. Classifier Chains

In [26]:
```python
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.linear_model import LogisticRegression
```

In [27]:
```python
%%time

# initialize classifier chains multi-label classifier
classifier = ClassifierChain(LogisticRegression())

# Training logistic regression model on train data
classifier.fit(x_train, y_train)

# predict
predictions = classifier.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("\n")
```

```
Accuracy =  0.8933333333333333


CPU times: user 6.2 s, sys: 2.52 s, total: 8.72 s
Wall time: 8.64 s
```

3.4. Label Powerset

In [28]:
```python
# using Label Powerset
from skmultilearn.problem_transform import LabelPowerset
```

In [29]:
```python
%%time

# initialize label powerset multi-label classifier
classifier = LabelPowerset(LogisticRegression())

# train
classifier.fit(x_train, y_train)

# predict
predictions = classifier.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("\n")
```

```
Accuracy =  0.8933333333333333


CPU times: user 550 ms, sys: 209 ms, total: 759 ms
Wall time: 662 ms
```

3.5. Adapted Algorithm

In [30]:

```
# http://scikit.ml/api/api/skmultilearn.adapt.html#skmultilearn.adapt.MLkNN

from skmultilearn.adapt import MLkNN
from scipy.sparse import csr_matrix, lil_matrix
```

In [31]:

```
%%time

classifier_new = MLkNN(k=10)

# Note that this classifier can throw up errors when handling sparse matrices.

x_train = lil_matrix(x_train).toarray()
y_train = lil_matrix(y_train).toarray()
x_test = lil_matrix(x_test).toarray()

# train
classifier_new.fit(x_train, y_train)

# predict
predictions_new = classifier_new.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions_new))
print("\n")
```

```
Accuracy =  0.8816666666666667


CPU times: user 2min 28s, sys: 839 ms, total: 2min 29s
Wall time: 2min 29s
```