Toxic Comment Classification

- Muddassir

Contents

Chapter 1

Introduction

1.1 Problem Statement

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

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 behaviors, 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).
We have to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate better than Perspective's current models. We'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.
This is Classification model. We have to predict toxicity of the each comment.

1.2 Data Description

We are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are:

- Toxic
- Severe Toxic
- Obscene
- Threat
- Insult
- Identity Hate

We must create a model which predicts a probability of each type of toxicity for each comment.

1.3 File Descriptions

Train.csv: the training set, contains comments with their binary labels
Test.csv: the test set, you must predict the toxicity probabilities for these comments. To deter hand labeling, the test set contains some comments which are not included in scoring.
Sample_submission.csv: a sample submission file in the correct format

Table 1.1: Training Data

	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

Predictors:

- Id
- Comment_text

Target Variables:

- Toxic
- Severe_toxic
- Obscene
- Threat
- Insult
- Identity_hate

Table 1.2: Test Data

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.

Table 1.3: Sample Submission File

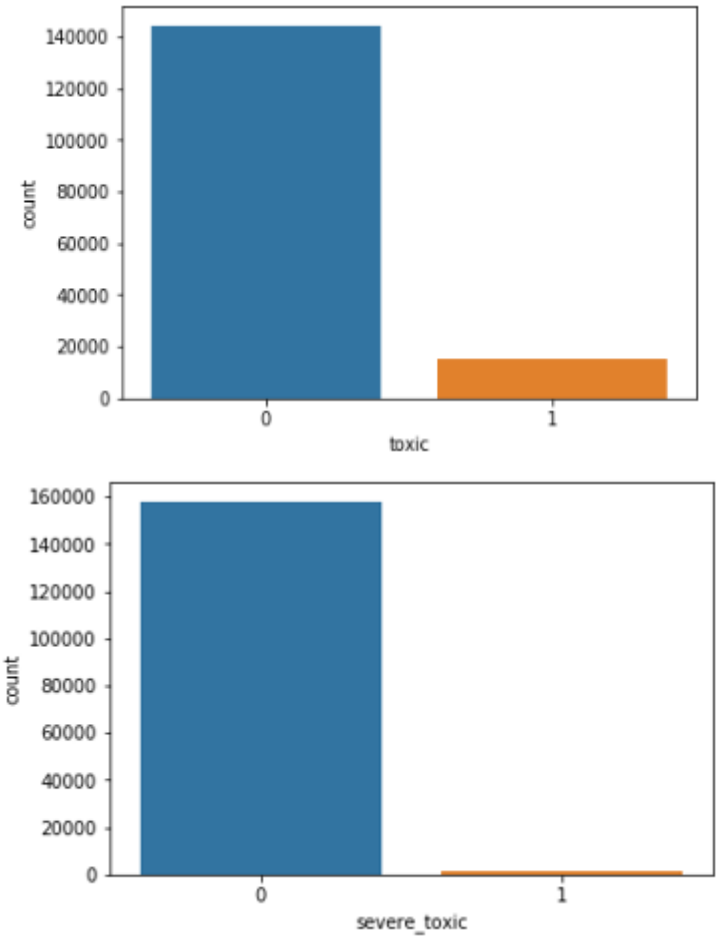
	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	0.5	0.5	0.5	0.5	0.5	0.5
1	0000247867823ef7	0.5	0.5	0.5	0.5	0.5	0.5
2	00013b17ad220c46	0.5	0.5	0.5	0.5	0.5	0.5
3	00017563c3f7919a	0.5	0.5	0.5	0.5	0.5	0.5
4	00017695ad8997eb	0.5	0.5	0.5	0.5	0.5	0.5

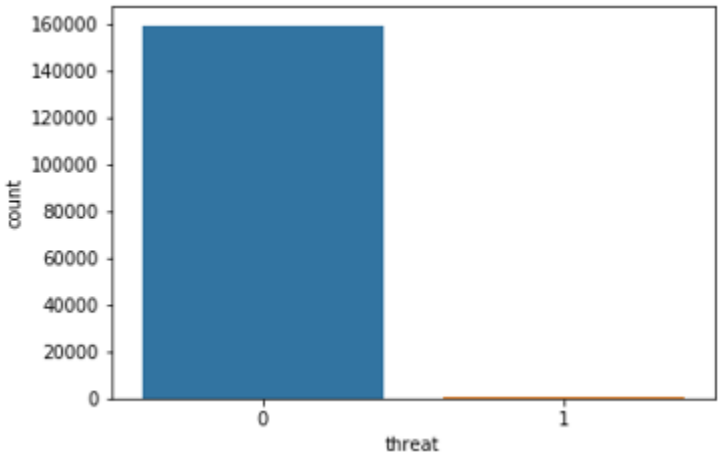
Chapter 2

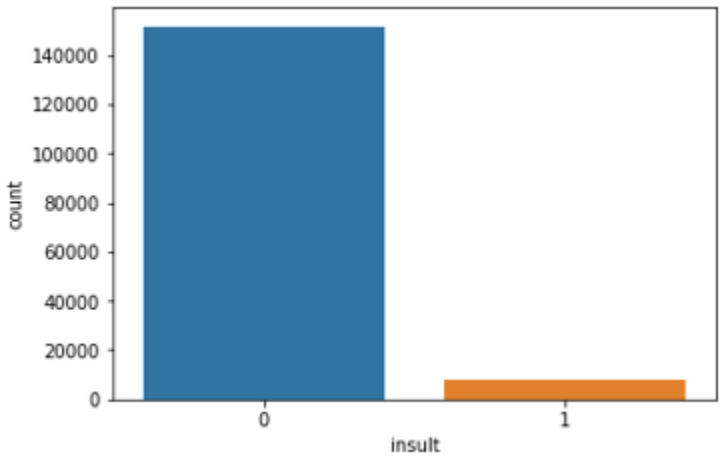
Exploratory Data Analysis

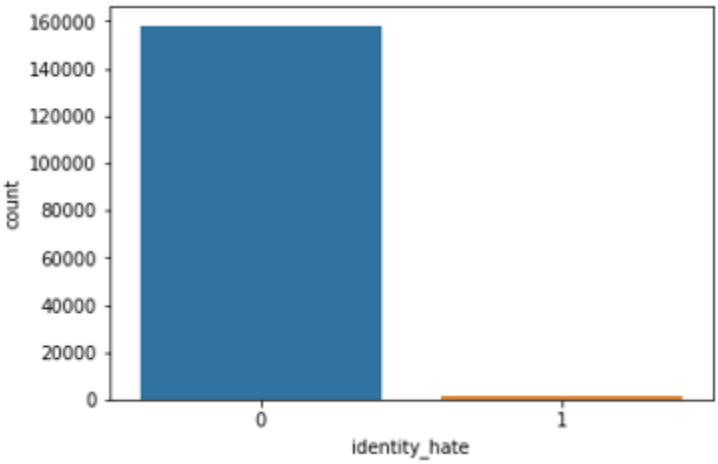
Exploratory Data Analysis (EDA) is the first step in our data analysis process. Here, we make sense of the data we have and then figure out what questions we want to ask and how to frame them, as well as how best to manipulate our available data sources to get the answers we need.

2.1 Checking the counts of Target variables through Histograms









We can see that, for each target variable, there are more clean comments than toxic comments.

2.2 Checking more Toxic Target Variable

```
In [82]: for col in train.columns[2:]:
             c=train[train[col]==1].iloc[:,2].count()
             print("{} count is {}".format(col,c))

toxic count is 15294
severe_toxic count is 1595
obscene count is 8449
threat count is 478
insult count is 7877
identity_hate count is 1405
```

We can see that Toxic variable has more toxic comments

2.3 Using Word Cloud

Word clouds are normally used to display the frequency of appearance of words in a particular document or speech

Most frequently used words appear larger in the word cloud.

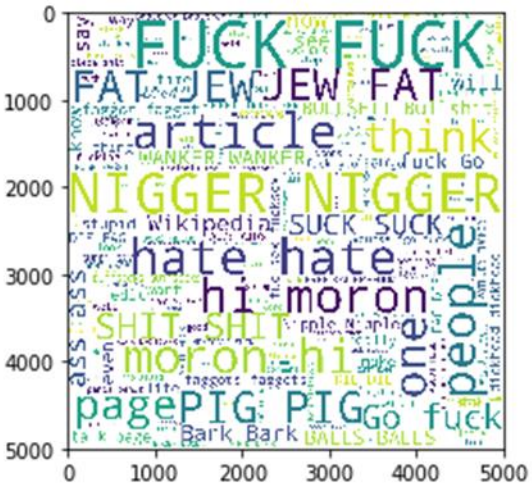
The frequency is assumed to reflect the importance of the term in the context of the document.

Frequently used words for “toxic” variable:

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
train=pd.read_csv("C:/ed/projects/Toxic Comment Classification/train.csv/train.csv")
temp=train[(train.iloc[:,2]==1)]
```

In [4]:
```
from wordcloud import WordCloud,STOPWORDS
wordcloud=WordCloud(width=5000,height=5000,stopwords=STOPWORDS,background_color='white').generate(" ".join(temp["comment_text"]))
import matplotlib.pyplot as plt
plt.figure(figsize=(15,8))
plt.axis('off')
plt.show()
plt.imshow(wordcloud)
```

Out[4]: <matplotlib.figure.AxesImage at 0x23446eab5f8>



Frequently used words for Severe_Toxic Variable:

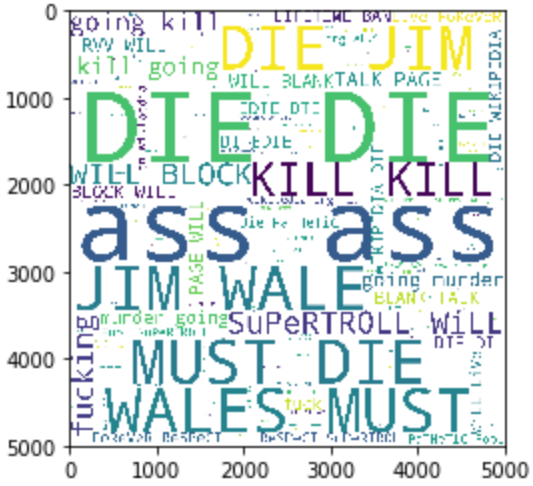
```
Out[6]: <matplotlib.image.AxesImage at 0x23456ab6208>
```

Frequently used words for “Obscene” variable:

```
Out[8]: <matplotlib.image.AxesImage at 0x234050b03c8>
```

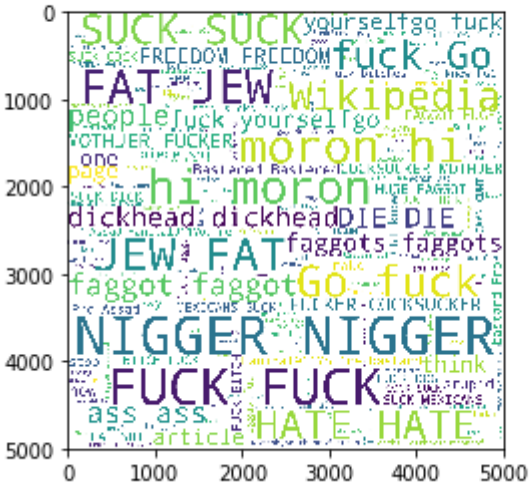
Frequently used word for “threat” variable:

```
Out[10]: <matplotlib.image.AxesImage at 0x23408e2b518>
```



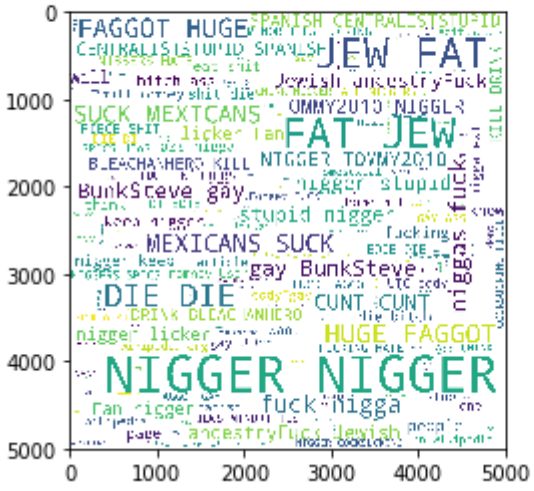
Frequently used word for “insult” variable:

```
Out[12]: <matplotlib.image.AxesImage at 0x23408fa18d0>
```



Frequently used word for “identity_hate” variable:

```
Out[14]: <matplotlib.image.AxesImage at 0x2345182af60>
```



Chapter 3

Pre-Processing

3.1 Missing Value Analysis

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analyzed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

```
In [11]: pd.isnull(train).sum()

Out[11]: id              0
         comment_text    0
         toxic           0
         severe_toxic    0
         obscene         0
         threat          0
         insult          0
         identity_hate   0
         dtype: int64
```

We can see that no column has missing values.

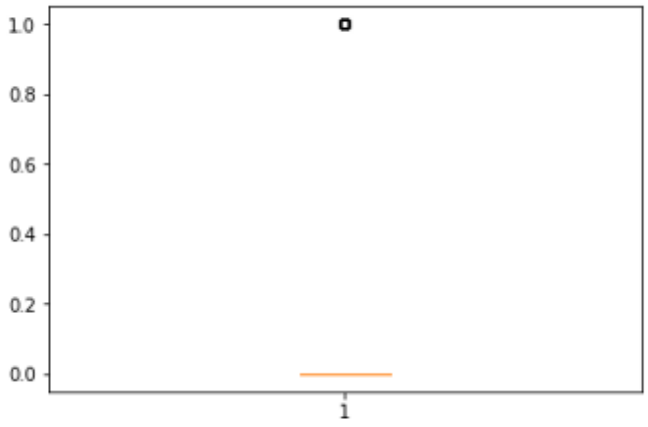
3.2 Outlier Analysis

Outlier is an observation that appears far away and diverges from an overall pattern in a sample. Outliers can drastically change the results of the data analysis and statistical modeling. There are numerous unfavorable impacts of outliers in the data set:

- It increases the error variance and reduces the power of statistical tests
- If the outliers are non-randomly distributed, they can decrease normality
- They can bias or influence estimates that may be of substantive interest
- They can also impact the basic assumption of Regression, ANOVA and other statistical model assumptions.

```
In [24]: plt.boxplot(train["identity_hate"])
```

```
Out[24]: {'boxes': [<matplotlib.lines.Line2D at 0x1e82f3a4400>],
          'caps': [<matplotlib.lines.Line2D at 0x1e82f3a4f28>,
           <matplotlib.lines.Line2D at 0x1e82f3af3c8>],
          'fliers': [<matplotlib.lines.Line2D at 0x1e82f3afc88>],
          'means': [],
          'medians': [<matplotlib.lines.Line2D at 0x1e82f3af828>],
          'whiskers': [<matplotlib.lines.Line2D at 0x1e82f3a4588>,
           <matplotlib.lines.Line2D at 0x1e82f3a4ac8>]}
```



By plotting boxplot for all variables,we can know that there are no values except ‘0’ or ‘1’

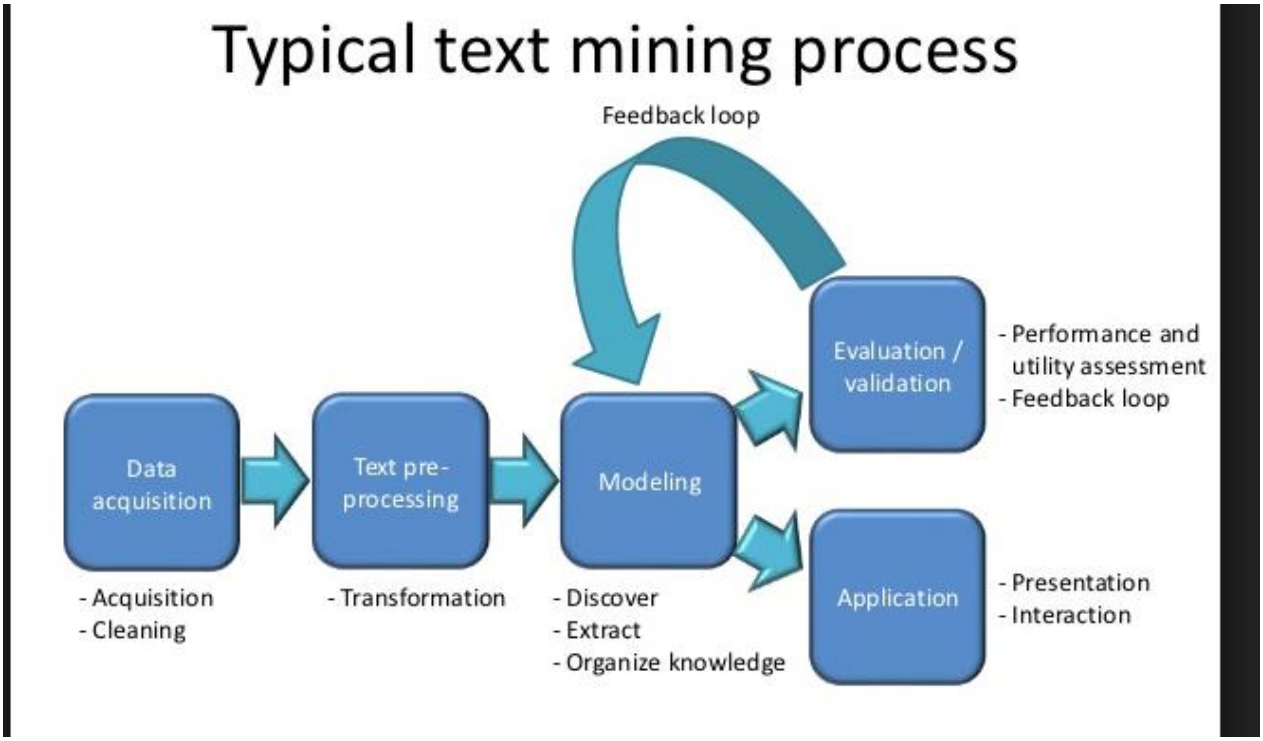
Chapter 4

Methodology

4.1 Text Mining

- Process of extracting interesting and non-trivial information and knowlede from unstructured text.
- Process of identifying novel information from a collection of texts(also known as corpus)
- Discover useful and previously unknown “gems” of information in large text collections

4.2 Process of Text Mining



4.3 Data Acquisition

Acquring data from different resources

Few sources of text:

- Facebook
- Twitter
- Yahoo
- Google
- Blogs
- Whatsapp

We need not acquire the data for this project.
Data is already extracted.

First step we have to import required libraries:

Importing libraries:

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
```

Second step is reading training and test data

Reading Training and test Data:

```
train=pd.read_csv("C:/Users/mudmoham/Desktop/train.csv/train.csv")
test=pd.read_csv("C:/Users/mudmoham/Desktop/test.csv/test.csv")
```

4.4 Pre-Processing

- Pre-Processing is simply transformation step.First we will remove unnecessary data.
- It is step before modeling.
- Algorithms only work on structured data.
- Algorithms should not contain unncessary data.
- In pre-processing step,first we will remove unnecessary data,then we will convert unstructrured to structured data

4.5 Removing Unnecessary data(converting to Text to Tokens)

- Punctuation marks
- Numbers
- Case Folding
- Stop Words
- White spaces
- Stemming
- Lemmatization
- Synonnym check

4.6 Unstructured Data to Structured Data

- A collection of n documents can be represented in the vector space model is called a Term Document Matrix
- An entry in the matrix corresponds to weight of the of a term in the document

TF-IDF Weighting:

A typical weighting is called tf-idf weighting.

W=tf*idf

Term Frequency:

More frequent terms in a document are more important

TF=(Number of times term t appears in a document) /(Total Number of terms in the document)

Inverse Document Frequency:

Terms that appear in many different documents are less important
IDF= log10(Total number of documents/Number of documents with term t in it)

```
tdm = TfidfVectorizer(analyzer='word',
                      token_pattern=r'\w{1,}',
                      stop_words='english',
                      max_features=5000)
```

In the above code we have seen that with the help of TfidfVectorizer we have done pre processing step and transformation in one step.
TfidfVectorizer converts data to “Term Document Matrix” i.e.. structrued data.
It will first remove unnecessary data like stop_words,punctuation,unnecesary spaces.
It wil do case folding,lemmatization automatically.
Then it will convert corpus into vector form i.e..Term document Matrix

	T1	T2	T3……..Tn
D1	W11	W12	W13……W1n
D2	W21	W22	W23……W2n
:	:	:	: :

Here weights are calculated by using formula:

W=TF*IDF

4.7 Modeling

Selecting the model:

We have iterate all models and select the model which fits the data best and which gives better accuracy.
To select the model,we make use of Cross validation score.

Cross-validation Technique:

Cross Validation is a technique which involves reserving a particular sample of a data set on which you do not train the model. Later, you test the model on this sample before finalizing the model.

Here are the steps involved in cross validation:

1. We *reserve* a sample data set.
2. Train the model using the remaining part of the data set.
3. Use the reserve sample of the data set test (validation) set. This will help us to know the effectiveness of model performance. It our model delivers a positive result on validation data, go ahead with current model.

k- Fold cross validation:

1. Randomly split our entire dataset into k ”folds”.
2. For each k folds in our dataset, build our model on k – 1 folds of the data set. Then, test the model to check the effectiveness for kth fold.
3. Record the error you see on each of the predictions.
4. Repeat this until each of the k folds has served as the test set.
5. The average of our k recorded errors is called the cross-validation error and will serve as our performance metric for the model.

Code for Calculating Cross Validation Score:

```
lr=LogisticRegression()

lr.fit(x_train,y_train)

cv_score=np.mean(cross_val_score(lr,x_train,y_train,cv=5,scoring='roc_auc'))
```

We can see that we have calculated cross validation score using logistic regression for training calculate cv scores for all other classification models.

Freeze the model which gives high cv score.

For this data set, Logistic Regression gives high cv score.

Hence we have freeze Logistic Regression Model.

Logistic Regression Model:

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.

Code to build the model:

```
from sklearn.linear_model import LogisticRegression

lr=LogisticRegression(C=5,solver='sag')
lr.fit(x_train,y_train)
```

4.8 Tuning the Parameters

Solver: This is the parameter used for optimization.
Solver='sag' which means we are using 'sag' algorithm for optimazation.
'sag' algorithm is best suitable for larger datasets

C Parameter:

- Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
- The trade-off parameter of logistic regression that determines the strength of the regularization is called C, and higher values of C correspond to less regularization (where we can specify the regularization function).C is actually the Inverse of regularization strength(lambda)

Code for choosing Best Value for "C" using GridSearchCV:

- The **GridsearchCV** instance implements the usual estimator API: when "fitting" it on a dataset all the possible combinations of parameter values are evaluated and the best combination is retained.

- Using GridSearchCV is easy. We just need to import GridSearchCV from sklearn.grid_search, setup a parameter grid and then pass the algorithm, parameter grid and number of cross validations to the GridSearchCV method.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

train = pd.read_csv('../input/train.csv').fillna(' ')
test = pd.read_csv('../input/test.csv').fillna(' ')

complete_text=pd.concat([train["comment_text"],test["comment_text"]])
col_names=train.columns[2:]
tdm = TfidfVectorizer(analyzer='word',
                      token_pattern=r'\w{1,}',
                      stop_words='english',
                      max_features=5000)

tdm.fit(complete_text)
train_tdm=tdm.transform(train["comment_text"])
test_tdm=tdm.transform(test["comment_text"])
x_train=train_tdm.toarray()
```

```
x_test=test_tdm.toarray()
y_train=train["toxic"].values
logistic=LogisticRegression(solver='sag')
C=[0.01,0.1,1,3,5,10,100]
hyperparameters=dict(C=C)
clf=GridSearchCV(logistic,param_grid=hyperparameters,cv=3)
best_model=clf.fit(x_train,y_train)
print('Best C:', best_model.best_estimator_.get_params()['C'])
```

Output:

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/sag.py:326: ConvergenceWarning: The
  "the coef_ did not converge", ConvergenceWarning)
Best C: 5
```

Inference:
Hence our Logistic Regression model will be best fit if we tune the paramater to C=5

4.9 Evaluation

```
cv_score=np.mean(cross_val_score(lr,x_train,y_train))
```

Evaluate the model again by using cross validation score with K fold cross validation technique with number of folds equal to 5.

Chapter 5

Complete Python Code

5.1 Python Code:

```
#Importing required libraries
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score

#Reading Trainig set and Test set.....Please replace the ur folder to in place".." to run
train = pd.read_csv('../input/train.csv').fillna(' ')
test = pd.read_csv('../input/test.csv').fillna(' ')
complete_text=pd.concat([train["comment_text"],test["comment_text"]])
col_names=train.columns[2:]


#Converting to Term Document Matrix with weight=TF*IDF

tdm = TfidfVectorizer(analyzer='word',
                      token_pattern=r'\w{1,}',
                      stop_words='english',
                      max_features=5000)

tdm.fit(complete_text)
train_tdm=tdm.transform(train["comment_text"])
test_tdm=tdm.transform(test["comment_text"])
x_train=train_tdm.toarray()
x_test=test_tdm.toarray()

#Declaring Dataframe for capturing output
output=pd.DataFrame(test["id"])
#list to store cross validation scores
```

```
scores=[]
```

#Applying Logistic model for all target columns

```
for col_name in col_names:
    y_train=train[col_name].values

    #logistic Regression
    lr=LogisticRegression()
    lr.fit(x_train,y_train)
    #cross validation scores
    cv_score=np.mean(cross_val_score(lr,x_train,y_train,scoring='roc_auc',cv=3))
    print("cv score for {} is {}".format(col_name,cv_score))
    scores.append(cv_score)
    output[col_name]=lr.predict_proba(x_test)[:,1]
```

#writing Output

```
output.to_csv("output.csv",index=False)
print("Total CV Score is {}".format(np.mean(scores)))
```

5.2 Output:

Code 0 additions, 0 deletions

Log
9 lines

Time	Line #	Log Message
2.6s	1	/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.DeprecationWarning: This module wa deprecated in version 0.18 in favo the model_selection module into wh all the refactored classes and fun are moved. Also note that the inte of the new CV iterators are differ from that of this module. This mod will be removed in 0.20. "This module will be removed in 0.20.", DeprecationWarning)
93.5s	2	cv score for toxic is 0.9644163513783598
127.1s	3	cv score for severe_toxic is 0.9847402983235268
160.6s	4	cv score for obscene is 0.9828650790961432
194.5s	5	cv score for threat is 0.9800278059889974
230.2s	6	cv score for insult is 0.9727807046161875
264.2s	7	cv score for identity_hate is 0.9710526388250774
267.4s	8	Total CV Score is 0.9759804797047154
269.2s	9	Total CV Score is 0.9759804797047154
269.2s	10	
269.2s	11	Complete. Exited with code 0.

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
output.csv	a few seconds ago	8 seconds	6 seconds	0.9723

Complete

5.3 Conclusion:

	A	B	C	D	E	F	G	H
1	id	toxic	severe_toxic	obscene	threat	insult	identity_hate	
2	00001cee341fdb12	0.993430444	0.177866483	0.993330981	0.028519596	0.8908117	0.191717507	
3	0000247867823ef7	0.00946091	0.002906144	0.004915479	0.001703103	0.008105496	0.003115467	
4	00013b17ad220c46	0.014773142	0.001517122	0.005997094	0.000798084	0.009606333	0.002333344	
5	00017563c3f7919a	0.004276442	0.002106206	0.003649595	0.001012306	0.004220549	0.000886656	
6	00017695ad8997eb	0.052140161	0.00181218	0.007682047	0.001770021	0.011728702	0.001753471	
7	0001ea8717f6de06	0.009703174	0.001621538	0.004480782	0.000818075	0.008712125	0.001611462	
8	00024115d4cbde0f	0.005756286	0.001256043	0.004620458	0.000538005	0.005335212	0.001067971	
9	000247e83dcc1211	0.516450384	0.003247269	0.054843278	0.003343609	0.1060107	0.005322957	
10	00025358d4737918	0.009985068	0.002431963	0.013450088	0.002363233	0.009228657	0.002389441	
11	00026d1092fe71cc	0.007681162	0.001128267	0.005599591	0.001013866	0.006203721	0.001783281	
12	0002eadc3b301559	0.377570376	0.001259192	0.109840478	0.001379385	0.02209511	0.002519602	
13	0002f87b16116a7f	0.049424828	0.002090875	0.019221267	0.001478868	0.010216729	0.004593058	
14	0003806b11932181	0.011383884	0.001467396	0.005865407	0.000571809	0.007282359	0.000630795	
15	0003e1cccfd5a40a	0.005480809	0.001663697	0.007085274	0.000916824	0.003974531	0.002127941	
16	00059ace3e3e9a53	0.004985122	0.000537289	0.002894746	0.000270333	0.003938451	0.000475686	
17	000634272d0d44eb	0.047698241	0.002498006	0.01336487	0.001709444	0.020458678	0.003456569	
18	000663aff0fffc80	0.060419969	0.005592695	0.018975708	0.002162024	0.022407317	0.005455801	
19	000689dd34e20979	0.03671607	0.001337772	0.010566103	0.00092036	0.013071962	0.003047137	
20	0008347691153 70c	0.003588428	0.000345038	0.001627423	0.000203434	0.00075857	0.000197566	
21	000844b52dee5f3f	0.034186245	0.002457003	0.010238477	0.000701225	0.015467213	0.001769137	
22	00084da5d4ead7aa	0.032392621	0.003636698	0.017973109	0.001251558	0.013633174	0.002931309	
23	00091c35fa9d0465	0.335041462	0.012583033	0.048073096	0.012886864	0.044220085	0.021248664	
24	000968ce11f5ee34	0.079498674	0.00287338	0.019142372	0.001267724	0.023706067	0.011090089	
25	0009734200a85047	0.008935989	0.002062314	0.008124444	0.001436934	0.005307825	0.003198925	
26	00097b6214686db5	0.026437664	0.002484643	0.012657833	0.002034027	0.010071695	0.005192214	
27	0009aef4bd9e1697	0.008804471	0.001724383	0.006124648	0.001292181	0.007404192	0.003101363	

We have successfully predicted Probabilites for all comments in the test file by applying Logistic Regression model where probabilities ranges from 0 to 1 with accuracy 97.2