CS4480 Data-Intensive Computing

Toxic Comments Modeling & Analysis

Lau Cho Shing (5481 4688)

Siu Man Kit (5454 0347)

Guo Kai Lun (5481 8675)

Chan Yat Chau (5485 1008)

11/30/2018

Abstract :

This project is produced by students in the City University of Hong Kong. Aimed to build predictive model from “Toxic Comment Dataset” which sourced from comments from Wikipedia’s talk page edits including 160 thousand comment records. Techniques including Stemming and Lemmatization have been adopted to clean and preprocess. Term Frequency–Inverse Document Frequency have been used to evaluate words importance. Three supervised learning models are builded including Logistic Regression, Decision Tree and Naive Bayes Classifier to predict the weather a comment is toxic and categorize the toxic type. The parallel computing framework - Spark have been adopted to speed up the computing process. Different methodologies are used for increasing the accuracy of the prediction.

Individual Contribution Statement:

Lau Cho Shing - Joe (5481 4688) 25%
(Background researching,Data preparation and reporting)

Siu Man Kit - Sam (5454 0347) 25%
(Data analysis and coding)

Guo Kai Lun - Charon (5481 8675) 25%
(Data analysis and coding)

Chan Yat Chau - Virgil (5485 1008) 25%
(Data preparation,coordination, reporting)

Table of content

Table of content 4

Introduction 4

Motivation 4

Objective 4

Data Sources 5

Methodology 6

Parallel Computering Environment 6

Apach Spark Parallel Computing Framework 6

Data Exploration & Visualization 7

Data Cleaning and Preprocessing 9

Porter Stemmer Algorithm 9

Lemmatization 10

TF-IDF 11

Model Building 12

Logistic Regression 12

Decision Tree 13

Naive Bayes 13

Results and Analysis 14

Prediction Result 14

Evaluation 16

Future improvement 18

Smote 18

Model Parameter tuning 18

Spark Cluster using Docker deploy on Kubernetes 19

Conclusion 4

Reference 21

Introduction

Nowadays, people are free to express their ideas and feeling because of right of speech. At the same time, laws are set to prevent people using language to attach or slander the others.

However, since there is no strict censorship or regulation for online platforms, people do not need to bear any responsibility for what they said in the Internet. As a result, people always say some rude or disrespectful words. This may make the others have bad feeling. In more serious situation, it is a kind of online bullying, people may suicide because of those toxic comments.

In this project, our team will using different technical approaches to analyze the internet comment text. Moreover, we also apply apache spark as the parallel computing environment in this project. The apache spark allow us set the cluster to increase the performance.

Motivation

As Computer Science students, we have the knowledge and ability to contribute for the purity of the environment in the Internet. We aim to improve online conversation by helping people to find out the disrespectful or rude words. Then, the Internet administrators can block or delete those unhealthy comments which can help to ensure online conversation environment clean and peaceful. In addition, it helps to prevent online bullying as well.

For the majority of new generations, when we encounter some problems, we always use the Internet for searching the answer. There are a lot of people who are willing to answer you. However, We may read some bad comments from the others. Actually, reading toxic comments is useless for problem solving. Removing such toxic comments can help us to make our work more effective.

Also, online shopping is very common recently. People can easily buy anything from Amazon, eBay and Taobao etc. However, there is one problem that we have no idea with the quality of the product. The only way to know more about the product is to read the product review. When people having unsatisfying shopping experience, they may making toxic comments more probably. Therefore, we may know the product quality by knowing the frequency of toxic reviews.

Objective

We will first filter or handle the raw data in the dataset which is already labeled as toxic comments. We aims to make it more systematic and well-organised. Next, we will analysis the feature of labeled toxic comments. Then, we will build a predictive model by using different methodologies. We will apply them to predict whether the comments are toxic or not in another dataset. In addition, We may also analysis which methodologies are the most accurate for the prediction.

Data Sources

We obtain our dataset from Kaggle(https://www.kaggle.com/). It is a website which contains a lot of datasets for science project. The dataset we used is from a Featured prediction Competition --Toxic Comment Classification Challenge (Identify and classify toxic online comments)

For this project, we just use two datasets “train_dataset”and “test_dataset” from the competition. Each of them has around 153000 records. The first one with the types of toxicity rated as labels for each comment is “train_dataset”. It is used for evaluating the model. We used 70% records as our training data and 30% records as our testing data. For this dataset, there are 8 attributes, which including “id”, “comment_text”, “toxic”, “obscene”, “threat’, “insult”, and “identit_hate”. There will be a “1” under the column of that type of toxicity if the comment is rated as toxic or there will be a “0” if the comment is not toxic.

	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

There is one more dataset “test_dataset” which contain only “id” and “comment toxic”. We will use it to predict the result to show whether the comment is toxic or not.

```
test.head()
```

	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.

For our project, as the column “id” refers to the unique number of each comment and it is useless for our model. We are not going to use it in this project. While the column “comment_text” will be the input used in the model. The data type of the model input is textual data. For the other columns, which are “toxic”, “obscene”, “threat”, “insult” and “identity_hate”, will be the model prediction output. The data type of the output attribute is the binary data.

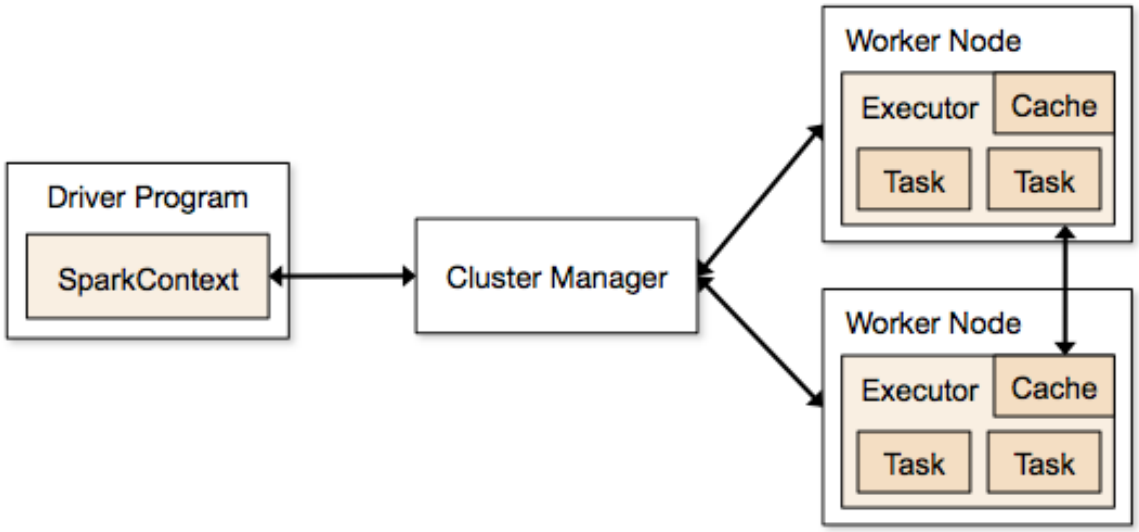
Finally, we can apply our model by using the the“comment_text” in the “test_dataset” for prediction result.

Methodology

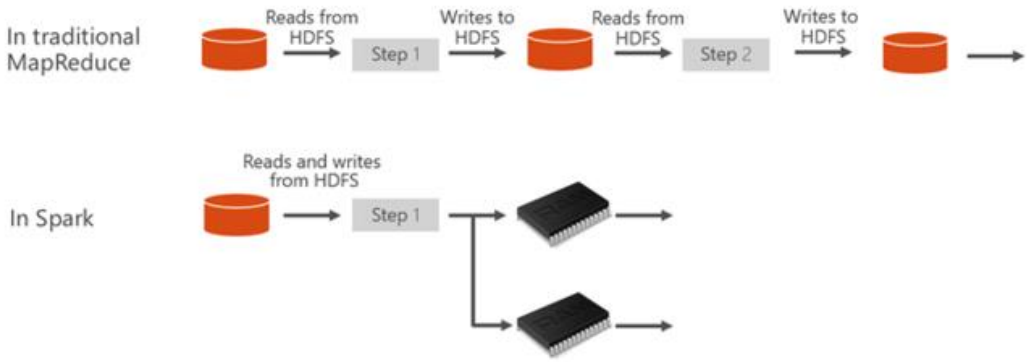
Parallel Computering Environment

Apach Spark Parallel Computing Framework

Spark is an distributed general-purpose cluster-computing framework. It allow the user using different cluster manager and distribute storage system. User can use the native Spark cluster, Hadoop Yarn, Apache Mesos, Kubemete and Amazon EC2 as the cluster manager (distributed computing part). For the distributed storage system, it allow user can use different enterprise storage solution as the storage, such as Amazon S3, Hadoop Distributed File System. Moreover, the spark framework is base on the mapreduce algorithm to build it up. it means the mapreduce function is one of spark basic component. In this project, our team config the native spark cluster as our cluster manager and Hadoop Distributed File System.



The above diagram is a simple spark clustering architecture of using the Apache Mesos as the cluster Management. One of using the spark clustering advantage is providing the dynamic resource allocation feature. Every tasks resource will release back to cluster once the job is no longer to use. The feature make the calculation performance increase when the resource is sharing with multi-process. Also, this feature can let worker node do not stay idle for too long time. This feature make our calculation performance increase because our team used different classifiers and different algorithms to evaluate and improve our model. it can make spend more time and rapid the project process.



The Apache spark Resilient Distributed Datasets also is one of spark framework advantage. it save intermediate data in RAM not in secondary storage. In the traditional MapReduce, it will load the data from HDFS before do the any calculation in every calculation and it will not store the data in RAM. Comparing the spark framework, the traditional MapReduce spend too many time in Read and Write to HDFS. For the spark, it only do Read and Writes from HDFS one time and all the processing data will be stored in memory after the user defined the dataset in code.

In [1]:
```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[4]") \
   .appName("ToxicCommentAnalysis") \
   .enableHiveSupport() \
   .getOrCreate()

sc = spark.sparkContext
```

Out project applied spark sudo cluster mode for parallel computing purpose. Instead of created a actual cluster and treat individual computer as worker node unit, this mode will treat each of the processor as worker and assign jobs to the unit. A single computer is used as the master node which responsible to assign and manage the jobs allocation and 4 of CPU core has assign to the cluster as worker node to receive the jobs assigned from master.

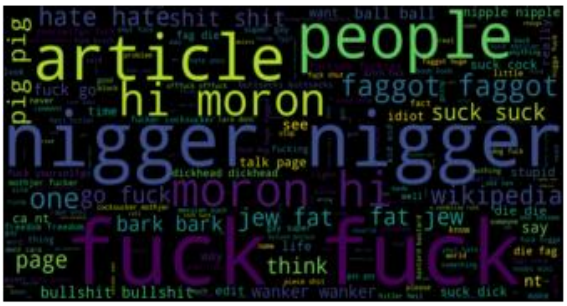
Under sudo cluster mode, data transfer between worker node, which is CPU cores, are fast when comparing to real cluster environment which using network. By shared the same piece of memory, data synchronization and communication is no longer required because of the shared memory model, any update of memory data will be notice by all other node.

With the RDD data structure, our input test file will be devidedinto four part and can be process simutainiusly.

Data Exploration & Visualization

As the input data for our model is textual data, we use word clouds to present the data in six types of toxicity. Word cloud is a visualization of word frequency in a given text as a weighted list.Here are the word clouds:

Toxic:

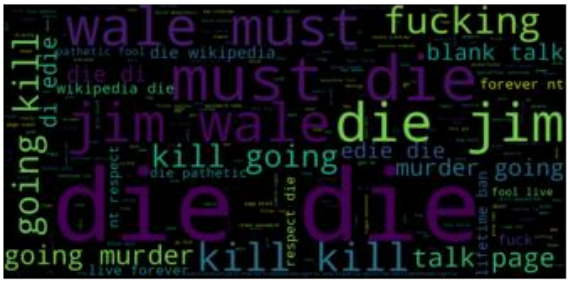


Servere_Toxic:

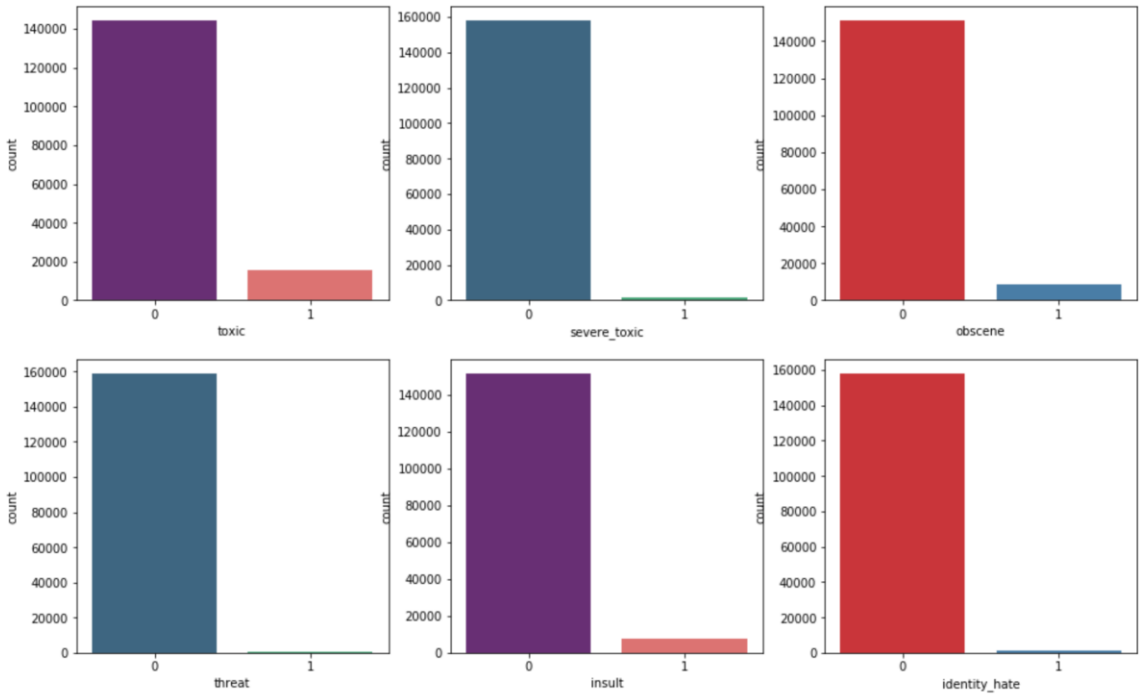
Obscene

Identity_hate

Threat:



Insult:



Data Cleaning and Preprocessing

Porter Stemmer Algorithm

The Porter Stemmer Algorithm is text processing algorithm that transform the word with remove the commoner morphological and text ending. Also, the algorithm will transform the word to lowercase. Actually, the algorithm has 4 main steps. They are including the below steps:

1. tokenization (transform the sentense to text vector)
2. filter out the stopwords
3. transform the word to lowercase
4. normalization the word (removing the word ending with Porter Stemmer Algorithm rule)

the For English language, a word can appear in different inflected forms or having derivational affixes. For example, the verb " to play" may appears as "playing", "played", "plays" and "play". In addition, there are some derivationally related words with similar meanings, such as "document", "documentation", "documentary" and "documental".Since the data type of "comment_text" attribute is textual data in the dataset, so we have to deal with the data first.

In this project, it having such different forms of a word is very difficult for us to do feature engineering. Moreover, the algorithm will ignore the word with numerics such as "apple19.123". therefore, we also filter out the numeric first before apply this algorithm. The step will change the word from "apple19.123" to "apple" and that can keep more information of the input text. In order to reduce inflectional forms and sometimes derivationally related forms of a word, we apply the Porter Stemmer Algorithm in this project. The Porter Stemmer Algorithm is very simple to understand. At most of the time, it can achieve the goal correctly by cutting off the ends of words, and often removing the derivational affixes.

```
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer

# print(string.punctuation)
def mytoken_stem(text):
    tokens = word_tokenize(text)
    tokens = [w.lower() for w in tokens]
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    words = [word for word in stripped if word.isalpha()]
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    stems = []
    for item in words:
        stems.append(PorterStemmer().stem(item))
    return stems
```

```
# tokenize
train['comment_text'] = train['comment_text'].map(lambda com : mytoken_stem(com))
```

By applying this algorithm, the example we just mention above can be easily handled. The words "playing", "played", "plays" and "play" will become "play". Also, the word "document" will replace the words "document", "documentation", "documentary" and "documental". It helps us to filter out the words with similar meaning which make us easier to analysis the features of the toxic comments.

Lemmatization

The Lemmatization is other methodology that reduce the grammatical form of word. it transform the different from to the basic form for word.
From the previous part, we just mentioned a word can appear in different inflected forms or having derivational affixes. Therefore, the Porter Stemmer Algorithm is used to chop off the ends of the words. However, this method is not good enough to handle some Irregular Verbs or some words in their derivationally related forms by having different part of speech.

For example, “seen”, “saw” and “see” have the same meaning with the verb “to see”. Also, The word “best” , “better” and “good” may refer to the positive meaning. If we just apply the Porter Stemmer Algorithm, they will be considered as three words. Therefore, we use one more method which is Lemmatization to deal with the textual data.

```
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer

# print(string.punctuation)
def mytoken_lemm(text):
    tokens = word_tokenize(text)
    tokens = [w.lower() for w in tokens]
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    words = [word for word in stripped if word.isalpha()]
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    lemms = []
    for item in words:
        lemms.append(WordNetLemmatizer().lemmatize(item))
    return lemms
```

```
# tokenize
train['comment_text'] = train['comment_text'].map(lambda com : mytoken_lemm(com))
```

Lemmatization is very similar to the Porter Stemmer Algorithm, however it is more powerful and well developed. It can help processing the word into their basic form, or in other name, lemma. The lemma for the word is the base form which can be looked up in a dictionary. For example, “worst”, and “worse” have “bad” as their example. This can help us to analysis the comments in a faster way.

TF-IDF

Term frequency–inverse document frequency are used to evaluate the importance of a certain word amount the documents. Term frequency is the number of occurrence of the given word within a given document and adjusted with the length of the document. Inverse Document frequency are calculated based on the number of document that the given word occured and adjusted by the total number of documents. The word that occur more in certain document but less amount all documents will be ranked higher in term of importance.

Term Frequency	$\log(1 + f_{t,d})$
Inverse Document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
Term frequency–Inverse document frequency	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$

Before TF-IDF, we first convert our tokenized comment into a numeric vector representing the occurrence of each words in comment.

This technique revealed the words that might significantly determine the meaning of comment records. By treating each of the comment as a document, the result of the method generated vector that rank the common word and stop words ,"the" "is" "at"..., lower. Which can reduce the interrupt effect of unimportant to the further classification result.

```
from pyspark.ml.feature import HashingTF,IDF,Tokenizer,RegexTokenizer,VectorAssembler

hashingTF = HashingTF(inputCol="comment_text", outputCol="rawCommentText", numFeatures=2000)
# featurizedData = hashingTF.transform(trainDF)
# featurizedData.dtypes

idf = IDF(inputCol="rawCommentText", outputCol="features")
# idfModel = idf.fit(featurizedData)
# finalTrainDF = idfModel.transform(featurizedData)
# finalTrainDF.dtypes
```

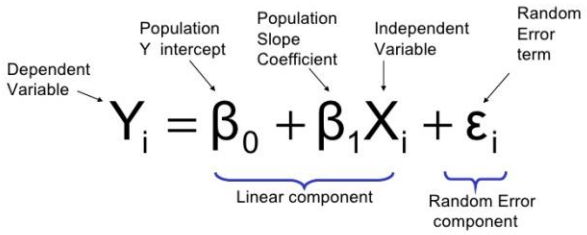
```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[hashingTF, idf])

# Fit the pipeline to training documents.
pipelineFit = pipeline.fit(trainDF)
trainingDF = pipelineFit.transform(trainDF)
```

```
# help(pipelineFit.save)
pipelineFit.write().overwrite().save('./models/tfidf-model')
```

Model Building

Logistic Regression

Logistic Regression is a classification model that map various weight input with resulting a categorical output. By taking multiple variables and its corresponding result, to calculate the coefficient of each variable to form the model. The trained model will perform a liner division of the sample space. It can take the variables to produce the estimate probabilities for prediction of weather a result will be ture or false for a given categories.



This model are used to predict the weather the occurrence of a sequence of words will result cause the comment categorise a given kind of toxic. Our model are trained by input the preprocessed weighted words vector generated from previous section, to calculate the coefficient of a given word in resulting of being label as a give toxic type. With the preprocess words vector, the weight of common word and stop word like,"the" "is" "at", that might not affect the meaning of comment will be low, and the weight of some rare word that might significantly affected the will be high. Combine with the computed coefficient from

logistic regression, the model can take the comment words vector to determine whether it belong to any categories of toxic comment by the result probabilities.

Decision Tree

Decision Tree is a nonlinear classification model which take sequence of input variable for predicting whether the result will fall in one of multiple categories. The model is to pick certain amount of variable based information gain and entropy to determine whether the variable should be applied to the prediction of result. The variable node will separate the sample data into two sub-set data. Entropy is used to measure impure level of the data set, which directed variate with the expected value of a certain result will be happened. The Information gain measure the decreased amount of entropy. This model will cause some variable might be neglected due to the chosen depth ,which is number of variable to use, of the tree model.

Information Gain

$$IG_{X,A}(X,a) = D_{\mathrm{KL}}(P_X(x|a) \| P_X(x|I)),$$

Entropy

$$\mathrm{H}(X) = \sum_{i=1}^{n} \mathrm{P}(x_i)\,\mathrm{I}(x_i) = -\sum_{i=1}^{n} \mathrm{P}(x_i)\log_b \mathrm{P}(x_i),$$

This model are used to find out the most influential variables amount all variable to predict whether the comment belong to a toxic category. It take the words vector and depth of tree as input. We can obtain the given words influence level by observing the depth of a chosen word. The result of this model will be automatically performed the feature selection and filtering which will be helpful to understand the model.

Naive Bayes

Naive Bayes is based on bayesian probability method, which representing the estimated conditional probabilities of a number of given variables. It is assumed that each of the input variables are mutually independent.

Naive Bayes

$$p(C_k \mid x_1, \ldots, x_n)$$

This model are used to find out the possibility of the comment that belong to any one of toxic category. It take the words vector that representing whether a words presented or not to calculate the likelihood of whether a certain result will occur.

Results and Analysis

Prediction Result

As we mentioned in the previous part Data Sources, we use the “test_dataset” as our model input. It only contain column “id” and “comment text”. After data-preprocess, we apply Stemming and Lemmatization to make the comments more easy for analysis. The data preprocessing result is as below:

```
test.head()
```

	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.

Before

```
test.head()
```

	id	comment_text
0	00001cee341fdb12	[yo, bitch, ja, rule, succesful, ever, whats, ...
1	0000247867823ef7	[rfc, title, fine, imo]
2	00013b17ad220c46	[source, zawe, ashton, lapland]
3	00017563c3f7919a	[look, back, source, information, updated, cor...
4	00017695ad8997eb	[nt, anonymously, edit, article]

After

Also, we have built three models by using three classifiers. All of them can successfully predict whether the comment is toxic or not (i.e. “0 “ for non-toxic, “1” for toxic). Also, it shows the probability of the comment being to be toxic. However, the three models have different results in prediction.

For Logistic Regression:

```
Predictions = lrModel.transform(testData)
Predictions.filter(Predictions['prediction'] == 1) \
    .select("toxic","probability","comment_text","prediction") \
    .orderBy("probability", ascending=False) \
    .show(n = 50, truncate = 40)
```

```
+-----+----------------------------------------+----------------------------------------+----------+
|toxic|                             probability|                            comment_text|prediction|
+-----+----------------------------------------+----------------------------------------+----------+
|    1|[0.48615247404523143,0.5138475259547686]|[gwernolugly, stillborn, molesting, d...|       1.0|
|    0|[0.46850262472232423,0.5314973752776758]|[bye, bye, bye, bye, bye, bye, bye, b...|       1.0|
|    0|[0.45935129086746523,0.5406487091325348]|[weezer, okay, weezer, okay, weezer, ...|       1.0|
|    0| [0.4281364628304904,0.5718635371695097]|[say, grasping, straw, say, denying, ...|       1.0|
|    1| [0.42156911075436804,0.578430889245632]|[bastard, proassad, rebel, aleppo, ce...|       1.0|
|    1|[0.42125789860390744,0.5787421013960926]|[mothjer, fucker, cocksucker, mothjer...|       1.0|
|    1| [0.36937839001297096,0.630621609987029]|[rodullandemu, coming, ya, jewish, fu...|       1.0|
|    1| [0.3454239771234863,0.6545760228765136]|[fuck, fucking, twat, dont, u, dare, ...|       1.0|
|    0|  [0.3312186055104005,0.66878139440896]|[wikipedia, opposite, straight, wikip...|       1.0|
|    1|[0.31935906364982347,0.6806409363501765]|[wikipedia, love, cock, wikipedia, lo...|       1.0|
|    1|[0.29707796316744256,0.7029220368325574]|[u, r, n, idiot, hahahahahahahahahaha...|       1.0|
|    0|[0.29291818951679044,0.7070818104832095]|[epic, fail, epic, fail, epic, fail, ...|       1.0|
|    1| [0.28306388544224803,0.716936114557752]|[go, fuck, yourselfgo, fuck, yourself...|       1.0|
|    1| [0.27749860901713197,0.722501390982868]|[faggot, faggot, faggot, faggot, fagg...|       1.0|
|    1| [0.2596341367568731,0.7403658632431269]|[mean, time, guess, got, song, dedica...|       1.0|
|    1|[0.23381890483144854,0.7661810951685515]|[thanks, watching, wiki, raid, thread...|       1.0|
|    1|[0.23028657876906053,0.7697134212309394]|[martyman, sad, little, man, penis, e...|       1.0|
|    0|[0.21321683946100123,0.7867831605389988]|[repeat, biznitch, repeat, biznitch, ...|       1.0|
|    1|[0.18714453097474446,0.8128554690252555]|[suxk, dick, failed, block, wikipedia...|       1.0|
|    1|[0.15676460014603277,0.8432353998539672]|[bastard, bastard, bastard, bastard, ...|       1.0|
|    1|[0.15228309237238838,0.8477169076276115]|[piece, shit, fuck, warning, fuck, mu...|       1.0|
|    1| [0.1379006694684403,0.8620993305315597]|[fuck, vuvuzelas, fuck, vuvuzelas, fu...|       1.0|
|    1|[0.13201351829279212,0.8679864817072078]|[eat, shit, die, bitch, as, nigga, ea...|       1.0|
|    1|[0.12987114578093767,0.8701288542190624]|[bad, admin, bad, admin, bad, admin, ...|       1.0|
```

For Decision Tree:

```
Predictions = dtModel.transform(testData)
Predictions.filter(Predictions['prediction'] == 1) \
    .select("toxic","probability","comment_text","prediction") \
    .orderBy("probability", ascending=False) \
    .show(n = 50, truncate = 40)
```

```
+-----+----------------------------------------+----------------------------------------+----------+
|toxic|                             probability|                            comment_text|prediction|
+-----+----------------------------------------+----------------------------------------+----------+
|    1|[0.19886363636363635,0.8011363636363636]|[baseball, bug, rude, fucking, tool, ...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[calling, bitch, talk, least, back, s...|       1.0|
|    0|[0.19886363636363635,0.8011363636363636]|[certain, block, probably, remain, lo...|       1.0|
|    0|[0.19886363636363635,0.8011363636363636]|[contd, scoundrel, block, padmalskhmi...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|   [could, fucking, leave, alone, bitch]|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[danielrigal, bitch, fucking, geek, a...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[dare, u, german, piece, dog, shit, d...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[faggot, quit, fucking, wit, mii, sht...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fc, bayern, roster, edits, footnote,...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, argie, loving, wanker, fuck...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, citation, http, fucking, ha...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, die, lol, rofl, joke, hahah...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, faggot, ur, fucking, faggot...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, huge, ego, fucking, geek, l...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, kidding, lay, cry, complete...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, moron, fucking, stupid, lit...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, mother, as, pussy, want, su...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, nolife, car, fetishist, sin...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, nt, anti, british, hate, be...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[fucking, retarded, treating, fly, mi...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[gaba, p, join, long, list, complete,...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[gc, directly, relevant, israel, stri...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[gee, minute, reverting, redirects, c...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[go, ahead, fucking, fascist, cunt, b...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[good, night, good, night, fucking, s...|       1.0|
|    1|[0.19886363636363635,0.8011363636363636]|[guy, got, get, real, ban, anyway, al...|       1.0|
```

For Naive bayes:

```
Predictions = nbModel.transform(testData)
Predictions.filter(Predictions['prediction'] == 1) \
    .select("toxic","probability","comment_text","prediction") \
    .orderBy("probability", ascending=False) \
    .show(n = 50, truncate = 40)
```

```
+-----+----------------------------------------+----------------------------------------+----------+
|toxic|                             probability|                            comment_text|prediction|
+-----+----------------------------------------+----------------------------------------+----------+
|    0|[0.49790532020769335,0.5020946797923066]|[opposition, took, arm, syria, le, ki...|       1.0|
|    0|  [0.497847130091386,0.5021528699086141]|[right, block, people, falsely, accus...|       1.0|
|    0|[0.49674501002301563,0.5032549899769843]|              [sockpuppet, leave, alone]|       1.0|
|    0|   [0.496243898382711,0.503756101617289]|[nicole, kidman, australian, nicole, ...|       1.0|
|    0| [0.4961469403840546,0.5038530596159454]|[woah, im, fron, spokane, im, liberal...|       1.0|
|    1| [0.49590157754736797,0.504098422452632]|[figure, aspergers, pedantic, little,...|       1.0|
|    0|[0.49534736421233755,0.5046526357876625]|                         [rtl, may, gmt]|       1.0|
|    1| [0.4949675804447062,0.5050324195552938]|[vandalism, stop, vandalising, page, ...|       1.0|
|    0| [0.49468861051658897,0.505311389483411]|               [peer, reviewpipe, organ]|       1.0|
|    0| [0.4937091371061304,0.5062908628938697]|          [user, talk, giano, iiarchive]|       1.0|
|    0|[0.49303611241948947,0.5069638875805105]|[spelling, flak, correct, spelling, u...|       1.0|
|    0| [0.4926299341350164,0.5073700658649837]|                     [protect, protected]|       1.0|
|    1| [0.4926090684931615,0.5073909315068386]|                            [care, block]|       1.0|
|    1| [0.4919832712815331,0.5080167287184669]|[yes, agree, editor, stupid, stubborn...|       1.0|
|    1| [0.4916699704063656,0.5083300295936345]|[informing, pink, floyd, fan, much, b...|       1.0|
|    1| [0.4912239961159064,0.5087760038840936]|         [blocked, ca, nt, respond, lie]|       1.0|
|    0|[0.49039469555738024,0.5096053044426199]|                   [talkback, cheer, hi]|       1.0|
|    0|[0.48822547647803927,0.5117745235219607]| [okay, whatever, like, going, win, war]|       1.0|
|    1| [0.4871562467226798,0.5128437532773202]|[think, correct, edits, come, house, ...|       1.0|
|    0|  [0.4863742041919729,0.513625795808027]|[hmmmmm, think, may, tipped, hand, re...|       1.0|
|    0| [0.4861736610304803,0.5138263389695196]|[penny, every, clown, world, billiona...|       1.0|
|    0| [0.4850290701722906,0.5149709298277094]|[america, saturday, morning, cartoon,...|       1.0|
|    0| [0.4839138660091724,0.5160861339908277]|                        [missing, fun]|       1.0|
|    1| [0.4833745060893567,0.5166254939106433]|[guy, worst, people, ever, never, wan...|       1.0|
|    1| [0.4814249626314978,0.5185750373685021]|          [fight, club, fk, yeeaaaaahh]|       1.0|
|    0| [0.4812002232875894,0.5187997767124106]|[continue, vandalize, report, adminis...|       1.0|
```

Evaluation

Since the result are different among three models and the classes are very imbalanced between toxic comment and non-toxic comment. We will compare their Accuracy, Receiver Operating Characteristic(ROC) Curve, and Precision Recall(PR) Curve for making comparison.

ROC curves are used to see the model can distinguish between the true positives and negatives. Area under the ROC curve(AUC) which measures the entire two-dimensional area underneath the entire ROC curve.

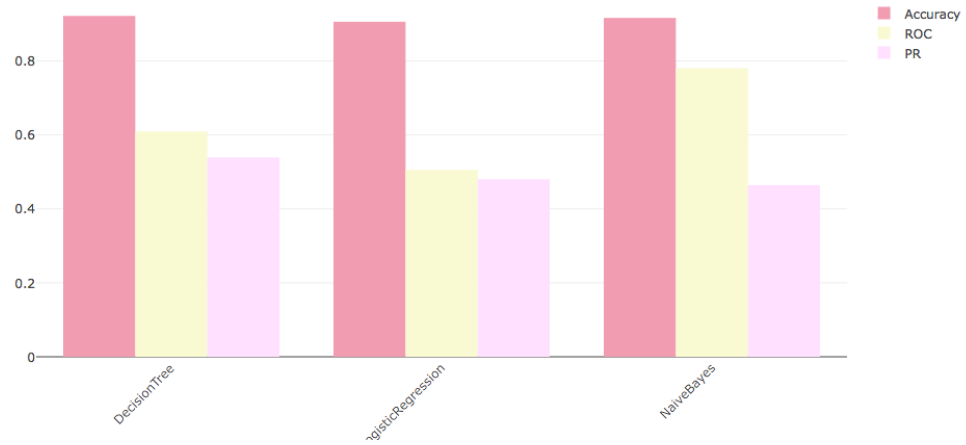
Precision-Recall curve is used to measure the success of prediction.
Precision is a measure for the relevancy of the result, while Recall is measuring the amount of returned truly relevant results.
For different threshold, the tradeoff between precision and recall will be shown as Precision-Recall curve

```
# Naive Bayes Evalution
from pyspark.ml.evaluation import MulticlassClassificationEvaluator,BinaryClassificationEvaluator


evaluator = MulticlassClassificationEvaluator(labelCol="toxic", predictionCol="prediction", metricName="accuracy")
nb_accuracy = evaluator.evaluate(Predictions)
print("Test Error = %g" % (1.0 - nb_accuracy))
print("Test Accuracy = %g" % nb_accuracy)

evaluator = BinaryClassificationEvaluator(rawPredictionCol='prediction', labelCol='toxic', metricName='areaUnderROC')
nb_roc = evaluator.evaluate(Predictions)
print("Test ROC = %g" %nb_roc)
nb_pr = evaluator.evaluate(Predictions, {evaluator.metricName: "areaUnderPR"})
print("Test PR = %g" %nb_pr)
```

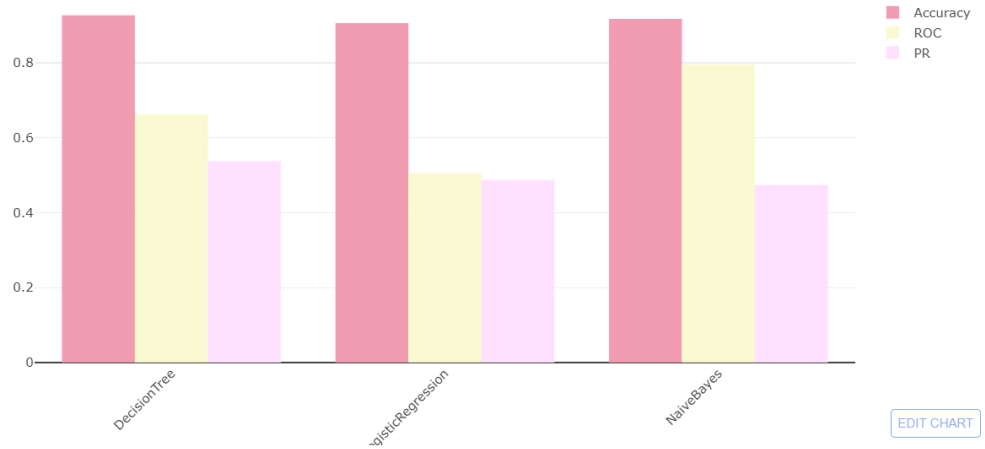
```
Test Error = 0.0836353
Test Accuracy = 0.916365
Test ROC = 0.78003
Test PR = 0.464262
```

Naive Bayes is the best model among three models, which has highest Accuracy and AUC.

The evaluation of Lammalization



The evaluation of Poter Stemming

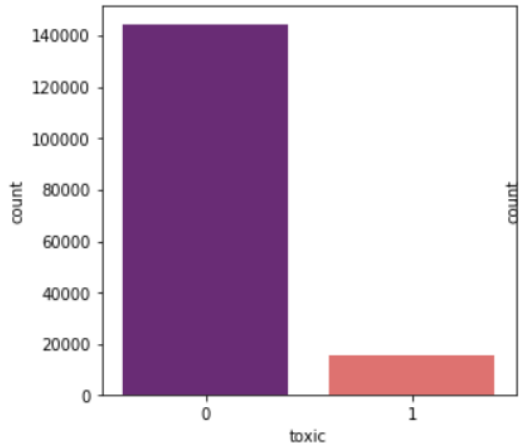


As can be seen from above, no matter compare through accuracy or AUC, the performance runs based on Porter stemming is slightly better than Lammalization.

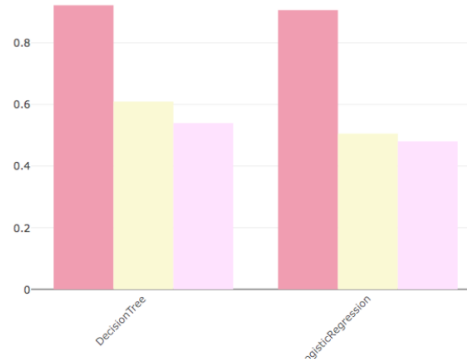
Future improvement

Smote

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally. The chart below shows our label size is extremely imbalanced. Around 85% of data is non-toxic and 15% of data is toxic.



The imbalanced label size cause the low AUC which shows below. The AUC is only around 50% - 60%.



Therefore, to fix the problem of imbalanced data, we will try to generate synthetic samples, one of the simplest way to achieve that is use SMOTE algorithm.

As its name suggests, SMOTE is an oversampling method. It works by creating synthetic samples from the minor class instead of creating copies. The algorithm selects two or more similar instances (using a distance measure) and perturbing an instance one attribute at a time by a random amount within the difference to the neighboring instances.

```
from imblearn.over_sampling import SMOTE
smo = SMOTE(random_state=42)
X_smo, y_smo = smo.fit_sample(trainingDF.select("features").collect(), trainingDF.select("toxic").collect())
```

Model Parameter tuning

Model's parameters are the variables that the chosen machine learning technique uses to adjust to the data. So keep tuning the model's variables is quite important. The same machine learning algorithm with different parameters could build different models. For example, in the Decision Tree, the number of trees could

Hyperparameters are the variables that govern the training process itself. In the model building period, we need to implement different models and different hyperparameters to build the different features, thus we need to search the hyperparameter space for the optimum values. The hyperparameter tuning methods includes Grid Search, Random Search, Smart Search and so on.

Spark Cluster using Docker deploy on Kubernetes

```
hashingTF = HashingTF(inputCol="comment_text", outputCol="rawCommentText", numFeatures=2000)
```

In our project, due to the large amount of features, the exception occurs when running the Random Forest algorithm and then lead the kernel crash. If we could use a real spark cluster, then the random forest model can be handled.

Random Forest

```
# from pyspark.ml.classification import RandomForestClassifier
# import time

# startTime = time.time()
# rf = RandomForestClassifier(labelCol="toxic", featuresCol="features", numTrees=8)
# rfModel = rf.fit(trainingData)

# executeTime = time.time() - startTime
# print ('Training Time For Random Forest = %s' % str(executeTime) + ' Seconds\n')
```

Furthermore, since deploy a spark cluster is kind of complicated, so we would like to build the spark cluster using docker images with HDFS,YARN,HIVE inside. Docker images make it portable and build once for all. Deploy it on Kubernetes makes it easy to scale up.

```
Charon Guo@LAPTOP-MRULN83I MINGW64 /c/Program Files/Docker Toolbox
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS


                                                NAMES
0d92d51f697a        617472db4d29        "/bin/bash"         2 days ago          Up 2 days           0.0.0.0:18043->8042/
tcp, 0.0.0.0:51011->50011/tcp, 0.0.0.0:51021->50021/tcp

                                                hadoop-node2
3d18d96c793a        617472db4d29        "/bin/bash"         2 days ago          Up 2 days           0.0.0.0:18042->8042/
tcp, 0.0.0.0:51010->50010/tcp, 0.0.0.0:51020->50020/tcp

                                                hadoop-node1
435d218f15a3        617472db4d29        "/bin/bash"         2 days ago          Up 2 days           0.0.0.0:17077->7077/
tcp, 0.0.0.0:18032->8032/tcp, 0.0.0.0:18050->8050/tcp, 0.0.0.0:18081->8081/tcp, 0.0.0.0:18888->8888/tcp, 0.0.0.0:18900->
8900/tcp, 0.0.0.0:19000->9000/tcp, 0.0.0.0:11100->11000/tcp, 0.0.0.0:28080->18080/tcp, 0.0.0.0:29888->19888/tcp, 0.0.0.0
:51030->50030/tcp, 0.0.0.0:51070->50070/tcp   hadoop-maste
a00a2041c6e5        617472db4d29        "/bin/bash"         2 days ago          Up 2 days


                                                mysql
b2eb360e846d        617472db4d29        "/bin/bash"         2 days ago          Up 2 days


                                                hive
```

Conclusion

For Toxic Comments Modeling and Analysis, we have successfully obtain our objectives. We use different ways to handle the raw data. Also, we use three methods to built different models for making prediction which can help to predict whether a comment is toxic or not.

We aim to contribute our efforts for making a clean and pure online conversation platform. Since the models are easy to implement that users just need to input the textual data for

making prediction, we hope our model can be wisely used in different platforms or products. As a result, no more toxic comments are shown in any online conversation.

For this project, we not only apply what we learn from lessons, but also use some new skills.

Regarding the knowledge we learn in lessons, we use The Porter Stemmer Algorithm, Lammalization and TF-IDF for data preprocessing. For data visualization, we use word cloud to show what words are common in the toxic comments. Also, we apply three classifiers for model building in order to make our prediction more accurate.

And we also learn some new skills from doing this project. First of all, we try to use Spark as our Parallel Computing Framework. We also apply ROC curve,PR curve and Accuracy to compare our prediction results.

All in all, we are feeling deep pleasure and satisfaction as a result of applying our knowledge to make contributions to the real world.

Reference

1. Kaggle--Toxic-comment-classification-challenge : https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

2. The Natural Language Processing Group of Stanford University: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
3. 6 Easy Steps to Learn Naive Bayes Algorithm https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
4. Decision Tree - Classification https://www.saedsayad.com/decision_tree.htm
5. How to process textual data using TF-IDF in Python https://medium.freecodecamp.org/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3
6. 4 key advantages of using decision trees for predictive analytics http://www.simafore.com/blog/bid/62333/4-key-advantages-of-using-decision-trees-for-predictive-analytics

7. Generate synthetic example - SMOTE https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/
8. Apache Spark - Introduction https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm

--- END ---