Algorithmics for Data Mining

Deliverable 1: Cyberbullying Detection on Social Media

Ali Arabyarmohammadi March 2022

1 Introduction

Social media has evolved into a global medium of interaction in people's lives. In addition to having a good impact, technological advancements have also generated new difficulties when they are misused. **Cybercrime** is a term used to describe this type of activity, and also **cyberbullying** is one type of Cybercrime that is popular right now. One of the media for the development of cyberbullying is social media, such as Twitter or YouTube.
The information was gathered from a variety of social media channels, including Twitter, Wikipedia Talk pages, and YouTube. The content which is present in the data, has been categorized as bullying or Normal. There are various sorts of cyber-bullying in the data, including hate speech, aggression, insults, and toxicity. Data mining techniques were used to conduct the study are Data collection, preprocessing, TF-IDF weighting, data validation, and classification using the Naive Bayes Classifier and some other methods that are all stages in this process.

In order to implement these mentioned data mining methods, **python** was used and the **sklearn** library was used which is a very helpful package in order to accomplish this study.

```
import pandas as pd
import numpy as np
import re
import nltk

from datetime import datetime
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
```

Listing 1: Python libraries were used

2 Data Understanding

Using **pandas** library, the dataset file **1_kaggle_parsed_dataset.csv** was loaded. The database is used adopted from the website **Mendeley** which is a collection of datasets relating to the automatic identification of cyber-bullying from various sources. In the table 1 the column "text" and the column "label" and some sample data are shown. The column "Label" represents if a text is considered bullying or not. Also, in the figure 1, number of bulling and normal data rows are shown.

```
df = pd.read_csv('1_kaggle_parsed_dataset.csv', sep="," , engine = 'c' ,encoding='
    latin1')
df.head(10)
```

ID	text	label
0	"You fuck your dad."	1
1	"i really don't understand your point.\xa0 It ...	0
2	"A\\xc2\\xa0majority of Canadians can and has ...	0
3	"listen if you dont wanna get married to a man...	0
4	"C\xe1c b\u1ea1n xu\u1ed1ng \u0111\u01b0\u1edd...	0
5	"@SDL OK, but I would hope they'd sign him to ...	0
6	"Yeah and where are you now?"	0
7	"shut the fuck up. you and the rest of your fa...	1
8	"Either you are fake or extremely stupid...may...	1
9	"That you are an idiot who understands neither...	1

Table 1: Dataset sample records

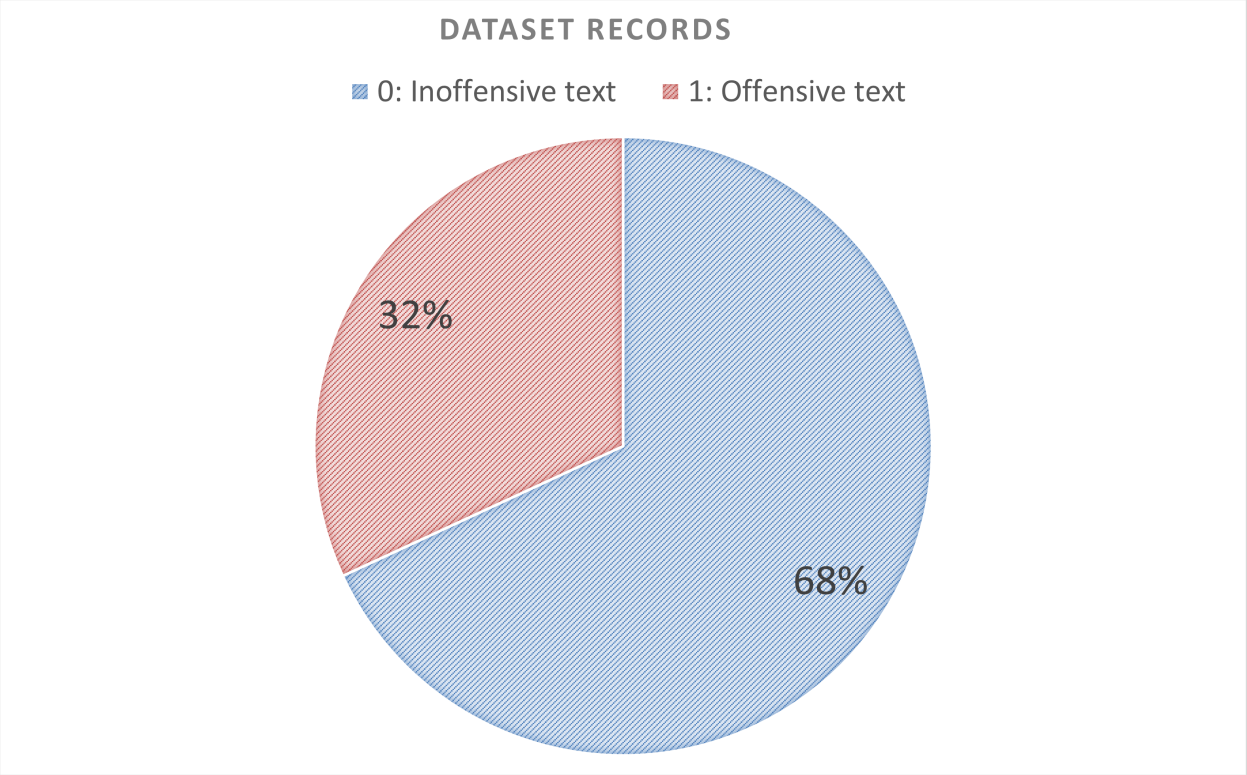


Figure 1: Pie chart showing number of bulling and normal data rows (Inoffensive text=5993 and Offensive text=2806)

2.1 Data Preparation

At the beginning of the code block below, blank rows will be removed, and all uppercase characters will be converted to the lower case to increase the precision. Then all words will be tokenized(line 8). The main Data pre-processing of the test is done inside the For-loop in the code below and precisely the stop words which were removed from the text (line 13). In the next few lines, numbers, single characters, special characters, commas, and multiple spaces.

```
start_time = datetime.now()
print('Start: {}'.format(start_time))

# Step - a : Remove blank rows if any.
Corpus['text'].dropna(inplace=True)
Corpus['label'].dropna(inplace=True)
Corpus['text'] = [entry.lower() for entry in Corpus['text']]
Corpus['text']= [word_tokenize(entry) for entry in Corpus['text']]

for index,entry in enumerate(Corpus['text']):
    Final_words = []
    for word in entry:
        if word not in stopwords.words('english'):
            word = re.sub("\d+$", " ", word)
            word =  re.sub(r"\b[a-zA-Z]\b", "", word) # Reomve single Chars: "this is a test" -> 'this is  test'
            word = re.sub('[^ A-Za-z0-9]+', '', word) # Remove Special Chars: "this is @ test" -> "this is  test"
            word = re.sub(r',', ' ', word) # Remove ','
            word = re.sub(' +', ' ', word) # Remove multiple spaces: 'this     is  a  test' -> 'this is a test'

            Final_words.append(word)

    Corpus.loc [index,'text'] = str(Final_words)
end_time = datetime.now()
print('Duration: {}'.format(end_time - start_time))
```

After the pre-processing stage, the data is divided into two parts which are Train and Test parts. In this study 20 percent is allocated to the Test and 80 percent goes for the Train part.

```
Train_X, Test_X, Train_Y, Test_Y = model_selection.
train_test_split(Corpus['text'],Corpus['label'],test_size=0.2)

Encoder = LabelEncoder()
Train_Y = Encoder.fit_transform(Train_Y)
Test_Y = Encoder.fit_transform(Test_Y)
```

After using the **encoder**, the train data would be in the format that is shown in the figure 2.

```
Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(Corpus['text'],Corpus['label'],test_size=0.2)
```

```
Train_Y
```

```
7638    0
4980    0
0       1
7408    0
8292    1
       ..
350     1
79      1
8039    0
6936    0
5640    1
Name: label, Length: 7039, dtype: int64
```

```
Encoder = LabelEncoder()
Train_Y = Encoder.fit_transform(Train_Y)
Test_Y = Encoder.fit_transform(Test_Y)
```

```
Train_Y
```

```
array([0, 0, 1, ..., 0, 0, 1], dtype=int64)
```

Figure 2: Form of data before and after using data encoder (Shown just for Trian_Y)

Because working on the text directly is not possible, first, we will perform the text vectorization, and after that data is converted to the numerical format, mining would be possible. Using the **TfidfVectorizer** function in the code below (line 1), we will convert a set of raw documents into a TF-IDF feature matrix.

```
1 Tfidf_vect = TfidfVectorizer(max_features=1000, ngram_range=(1, 3))
2 Tfidf_vect.fit(Corpus['text'])
3 Train_X_Tfidf = Tfidf_vect.transform(Train_X)
4 Test_X_Tfidf = Tfidf_vect.transform(Test_X)
```

3 Modeling

In this project, some methods listed below were used to perform the text mining, and the results of each method were compared with others.

- **Naive Bayes**
  Naive Bayes classifiers are a subset of simple "probabilistic classifiers" based on Bayes' theorem and strong feature independence requirements.

- **Random Forest**
  Random forests, also known as random choice forests, are an ensemble learning method for classification, regression, and other tasks that works by building a large number of decision trees during training.

- **Logistic Regression**
  A supervised learning approach called logistic regression is used to predict a dependent categorical target variable.

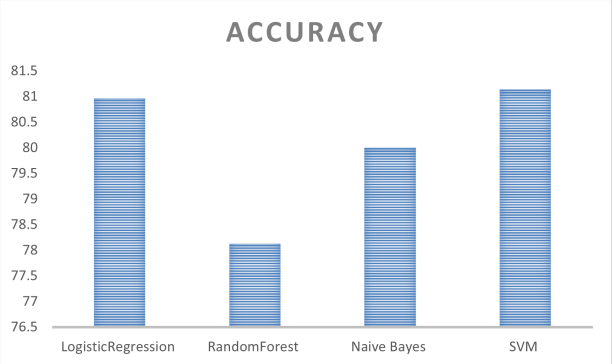
- **Support-Vector Machines (SVMs)**
  Support-vector machines are supervised learning models using learning algorithms that evaluate data for classification and regression analysis in machine learning.

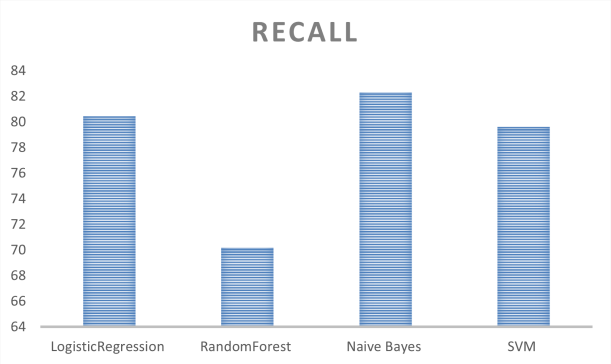
4 Results

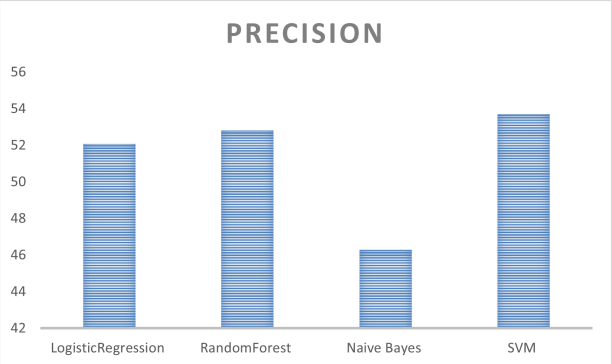
In the results section, they will be compared by considering some factors like **execution time, accuracy, recall, precision,** and **F-score**.

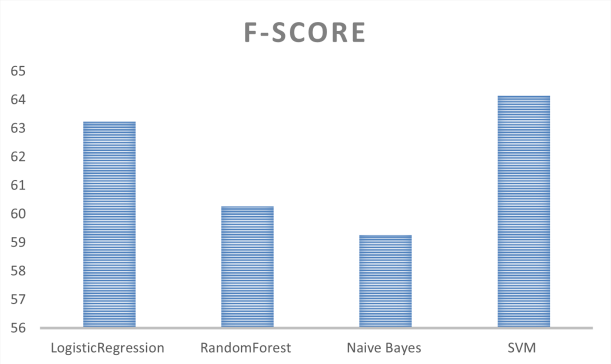
	Time	Accuracy	Recall	Precision	F-score
Logistic Regression	0:00:00.156999	80.96590909	80.44692737	52.079566	63.22722283
Random Forest	0:00:01.137983	78.125	70.19230769	52.80289331	60.26831785
Naive Bayes	0:00:00.007979	80	82.31511254	46.29294756	59.25925926
SVM	0:00:06.141998	81.13636364	79.62466488	53.70705244	64.14686825

Table 2: Results from four different methods









The best **execution time** in the Table 2 belongs to the Naive Bayes method and in the second level, Logistic Regression is located. In this column it is evident that SVM had the longest execution time of 6 seconds. In terms of **accuracy**, SVM performed better than the others with 81.13 percent but also the Logistic Regression is very close to it (80.9%). SVM has a higher **precision**, which is defined as the fraction of relevant examples among the recovered instances and the lower value is for the Naive Bayes method. Random Forest has the highest **recall**, which is the fraction of relevant documents that are successfully recovered, whereas Naive Bayes has the lowest. SVM has the highest **F-score**, which is computed from the test's precision and recall.

References

[1] Hariani, Imam Riadi, *Detection Of Cyberbullying On Social Media Using Data Mining Techniques* International Journal of Computer Science and Information Security (IJCSIS), Vol. 15, No. 3, March 2017.

[2] Eman Bashir, Mohamed Bouguessa, *Data Mining for Cyberbullying and Harassment Detection in Arabic Texts* I.J. Information Technology and Computer Science, 2021, 5, 41-50 - DOI: 10.5815/ijitcs.2021.05.04 - Published Online October 2021 in MECS.

[3] Amjad Rasmi Almutairi, Muhammad Abdullah Al-Hagery, *Cyberbullying Detection by Sentiment Analysis of Tweets' Contents Written in Arabic in Saudi Arabia Society* IJCSNS International Journal of Computer Science and Network Security, VOL.21 No.3, March 2021.