



**DEBRE BERHAN UNIVERSITY**

**College of Computing**

**Department – Software Engineering**

**Course title: Fundamental of Machine Learning**

**Course code: SEng9032**

**Group Assignment**

Group Member Names	Id_number
Melat Solomon	DBUR/1822/12
Samrawit Jarso	DBUR/0808/12
Nazrawit Gemechu	DBUR/1738/12
Ruth Abiti	DBUR/1873/12

## Abstract

This paper describes the system submitted by our team for the Group Assignment of the shared task “Hate Speech Detection” for the machine learning course we are taking currently at DBU. The data-set we used for this project is available following this link, [https://drive.google.com/drive/folders/1uQiyJ\\_mDlOCcecMw7C-JYUs9bGnVJ\\_j8](https://drive.google.com/drive/folders/1uQiyJ_mDlOCcecMw7C-JYUs9bGnVJ_j8). This data-set is used to train the machine using SVM and Logistic Regression machine learning algorithms, based on classification model. The datasets consist of twitter messages with three class labels “Hate\_Speech” , “offensive\_Language” and “Neither” which we later on categorized as “Hate-speech” and “no-hate Speech”. The machine is trained to classify such messages in these two categories. The best performing classification models developed are applied on test data-set. The model which gives the highest accuracy result on training data-set, was experimented to predict the labels of respective Test data-set.

**Keywords:** Support Vector Classifier, Logistic Regression, Tokenization and Vectorization

## Table of Content

1. Introduction .....	4
2. Statement of the Problem .....	4
3. Objectives.....	4
4. Methodology .....	5
5. Implementation.....	9
6. Conclusion.....	<b>Error! Bookmark not defined.</b>
7. Recommendation.....	<b>Error! Bookmark not defined.</b>
8. Reference.....	13

## **1. Introduction**

In recent years, hate speech has been increasing in-person and online communication. The social media as well as other online platforms are playing an extensive role in the breeding and spread of hateful content – eventually which leads to hate crime. For example, according to recent surveys, the rise in online hate speech content has resulted in hate crimes. However, the manual process to identify and remove hate speech content is labor-intensive and time consuming. Due to these concerns and widespread hate speech content on the internet, there is a strong motivation for automatic hate speech detection.

The automatic detection of hate speech is a challenging task due to disagreements on different hate speech definitions. Therefore, some content might be hateful to some individuals and not to others, based on their concerned definitions.

Therefore, this project contributes to solving this problem by building ML classifiers model on standard hate speech datasets.

## **2. Statement of the Problem**

If we look at the definition of hate speech, it is described as, public incitement to violence or hatred directed to groups or individuals on the basis of certain characteristics, including race, colour, religion, descent, and national or ethnic origin. Consequently, hate speech has clearly negative connotations, as is generally accepted. First, it should be noted that hate speech has both a short-term and a long-term impact on individuals.

Therefore the problem that we are trying to address is the current outspread of hate speech across different online communities and investing our time and skill into this project hoping it will create a much safer environment for those who engage in different online communities

## **3. Objectives**

### **Our Objectives**

- The main objective of this project is to develop an automated machine learning based approach for detecting hate speech and offensive language
- Automated detection corresponds to automated learning such as machine learning: supervised and unsupervised learning. We use a supervised learning method to detect hate.
- Classify text into two classes(like: hate and non-hate)
- Monitor the Spread of online hate-related speech

#### 4. Methodology

In this project supervised machine learning is used by experimenting on two Classifiers, Logistic Regression and SVM. The labeled datasets were preprocessed for removal of noisy elements from its contents.

Appropriate features are extracted to enable the machine to learn offensive term patterns. Finally the performances of different Classifiers and feature models are compared to choose the best performing model.

#### Hate Speech Dataset

The data-set were available in English languages following the link, [https://drive.google.com/drive/folders/1uQiyJ\\_mDI0CcecMw7C-JYUs9bGnVJ\\_j8](https://drive.google.com/drive/folders/1uQiyJ_mDI0CcecMw7C-JYUs9bGnVJ_j8). Both the training datasets have 8 columns: first is ID, second is the count of the identified words of the specified classes, third is the number of hate speech words, fourth is the number of offensive language words, fifth is the number of words identified as neither, sixth is the label or the class of the text which is 2,1,0 which is neither, offensive language and hate respectively and the last column is the text/tweet to be labeled.

The data-set consists of 24,582 data in total where 20,582 are categorized as hate speech and 4161 are categorized as non-hate

Language	Type of text	%	Total
English	Hate	20582( 83.7% )	24582
	Non-Hate	4161( 16.9% )	

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

## Date Preprocessing

- The first thing we did was mapping the offensive language and the hate speech as hate speech and the niether class as non- hate speech

```
data['label'] = data['class'].map({0:"Hate Speech Detected",1:"Hate Speech Detected",2:"No Hate speech"})
data.head()
```

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet	label
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...	No Hate speech
1	1	3	0	3	0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...	Hate Speech Detected
2	2	3	0	3	0	1	!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...	Hate Speech Detected
3	3	3	0	2	1	1	!!!!!!! RT @C_G_Anderson: @viva_based she lo...	Hate Speech Detected
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...	Hate Speech Detected

- What are the challenges in dealing with tweets, since our dataset is a twitter dataset:
  - Inconsistent use of grammar, proper spelling, punctuation, and capitalization
  - Repetition of same character, and use of random abbreviations makes tokenization difficult
  - Use of special characters, i.e. emojis, emoticons, hashtags

Here we have defined the techniques used in preprocessing tweets:

- Letter casing: Converting all letters to either upper case or lower case.
- Tokenizing: Separating piece of text into tokens most often into individual words
- Lemmatization: Eliminating affixes ( suffixes, prefixes, infixes) from word in order to obtain word stem
- Vectorization: process of converting tokens to numbers since machine learning algorithm works with numbers and not text

below is the thought process involved with each of the specific steps we identified working with the dataset to prepare the data for the modeling process:

- We removed the hash from the hashtags
- We removed the HTML links since a lot of users link a website reference as part of the tweet.
- We then removed any punctuation.
- We then lowercased all the tweets for tokenizing.
- Additional steps before modeling includes tokenization, lemmatizing, stemming, and vectorizing.

### Data processing

```
# defining a function for data processing
def data_processing(data):
    data = data.lower()
    data = re.sub(r"https\S+|www\S+http\S+", '', data)
    data = re.sub(r'\@w+|\#', '', data)
    data = re.sub(r'^\w\s', '', data)
    return data
```

```
data.tweet = data['tweet'].apply(data_processing)
```

### Tokenization

```
def lemmatize(token):
    return WordNetLemmatizer().lemmatize(token, pos='v')

def tokenize(tweet):
    result = []
    for token in gensim.utils.simple_preprocess(tweet):
        if token not in gensim.parsing.preprocessing.STOPWORDS \
            and len(token) > 2:
            result.append(lemmatize(token))
    return result

def tokenize_and_lemmatize(df, col):
    df[col] = df[col].apply(lambda x: tokenize(x))
    df.tweet = df.tweet.apply(lambda x: str(x)[1:-1])

tokenize_and_lemmatize(data, 'tweet')
data.tweet.head(20)
```

## CountVectorization Vs TfidfVectorization

TfidfVectorizer and CountVectorizer both are methods for converting text data into vectors as model can process only numerical data.

In CountVectorizer we only count the number of times a word appears in the document which results in biasing in favour of most frequent words. this ends up in ignoring rare words which could have helped in processing our data more efficiently.

In TfidfVectorizer we consider overall document weightage of a word. It helps us in dealing with most frequent words. Using it we can penalize them. TfidfVectorizer weights the word counts by a measure of how often they appear in the documents

### Vectorization( CountVectorizer ), Splitting data in to Training and testing data, Training and Testing the Model ( Logistic Regression )

```
#model Using CountVectorizer
cv = CountVectorizer()
X=cv.fit_transform(data['tweet'])
Y=LabelEncoder().fit_transform(data['label'])
```

```
X_train,x_test,Y_train,y_test = train_test_split(X,Y,test_size=0.2)
```

```
model = LogisticRegression().fit(X_train,Y_train)
pred = model.predict(x_test)
model_accuracy = accuracy_score(pred,y_test)
print("Model Accuracy: {:.2f}%".format(model_accuracy*100))
```

Model Accuracy: 95.57%

### Vectorization( TfidfVectorizer ), Splitting data in to Training and testing data, Training and Testing the Model ( Logistic Regression )



```
# model using tfidfVectorizer
tfidf = TfidfVectorizer()
X_t=tfidf.fit_transform(data['tweet'])
Y_t=LabelEncoder().fit_transform(data['label'])
```

```
X_train_t,x_test_t,Y_train_t,y_test_t = train_test_split(X_t,Y_t,test_size=0.2)
```

```
model_t = LogisticRegression().fit(X_train_t,Y_train_t)
pred_t = model_t.predict(x_test_t)
model_accuracy = accuracy_score(pred_t,y_test_t)
print("Model Accuracy: {:.2f}%".format(model_accuracy*100))
```

Model Accuracy: 93.53%

## Vectorization( CountVectorizer ), Splitting data in to Training and testing data, Training and Testing the Model ( SVM )

```
cv = CountVectorizer()
X=cv.fit_transform(data['tweet'])
Y=LabelEncoder().fit_transform(data['label'])
```

```
X_train,x_test,Y_train,y_test = train_test_split(X,Y,test_size=0.2)
```

```
model=svm.SVC(kernel='linear').fit(X_train,Y_train)
pred = model.predict(x_test)
model_accuracy = accuracy_score(pred,y_test)
print("Model Accuracy: {:.2f}%".format(model_accuracy*100))
```

Model Accuracy: 95.09%

## Vectorization( CountVectorizer ), Splitting data in to Training and testing data, Training and Testing the Model ( SVM )

```
# model using tfidfVectorizer
tfidf = TfidfVectorizer()
X_t=tfidf.fit_transform(data['tweet'])
Y_t=LabelEncoder().fit_transform(data['label'])
```

```
X_train_t,x_test_t,Y_train_t,y_test_t = train_test_split(X_t,Y_t,test_size=0.2)
```

```
model_t =svm.SVC(kernel='linear').fit(X_train_t,Y_train_t)
pred_t = model_t.predict(x_test_t)
model_accuracy = accuracy_score(pred_t,y_test_t)
print("Model Accuracy: {:.2f}%".format(model_accuracy*100))
```

Model Accuracy: 95.53%

## 5. Implementation

### Accuracy Comparision

	CountVectorizer	TfidfVectorizer
--	-----------------	-----------------

<b>Logistic Regression Model Accuracy</b>	95.57%	93.53%
<b>SVM Model Accuracy</b>	95.09%	95.53%

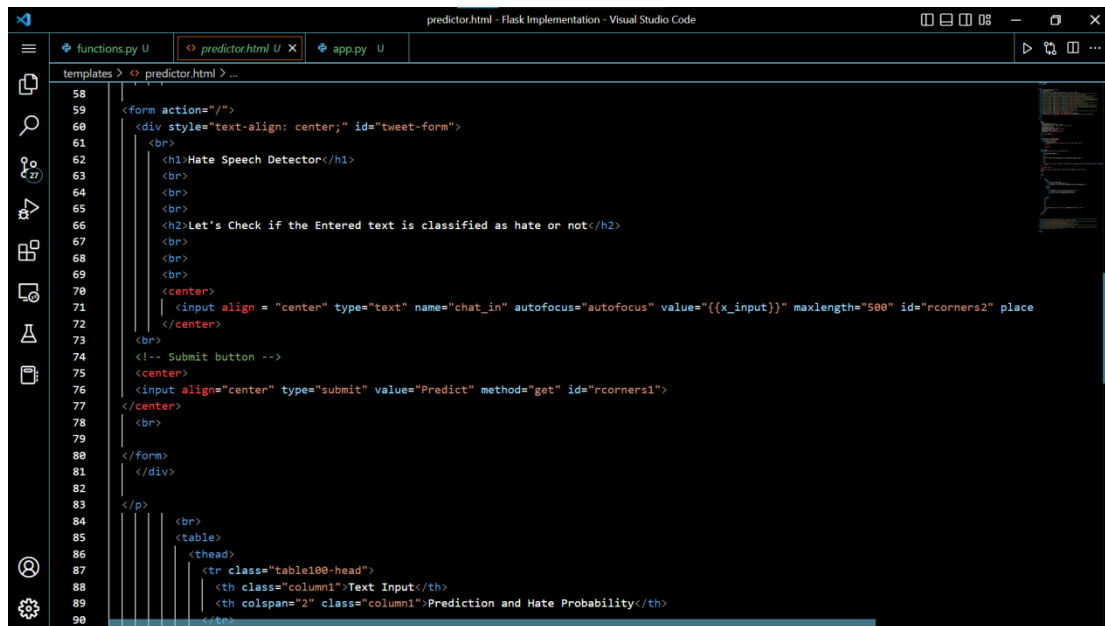
From the Accuracy we have deduced that its better to use the logistic Regression model using the CountVectorizer. In order to integrate the model to our interface, which is a website developed using the python framework Flask, we need to save it as a Pickle file is used for Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network.

### **Saving the Model**

```
pickle.dump(model,open('logistic Regression model.pkl','wb'))
```

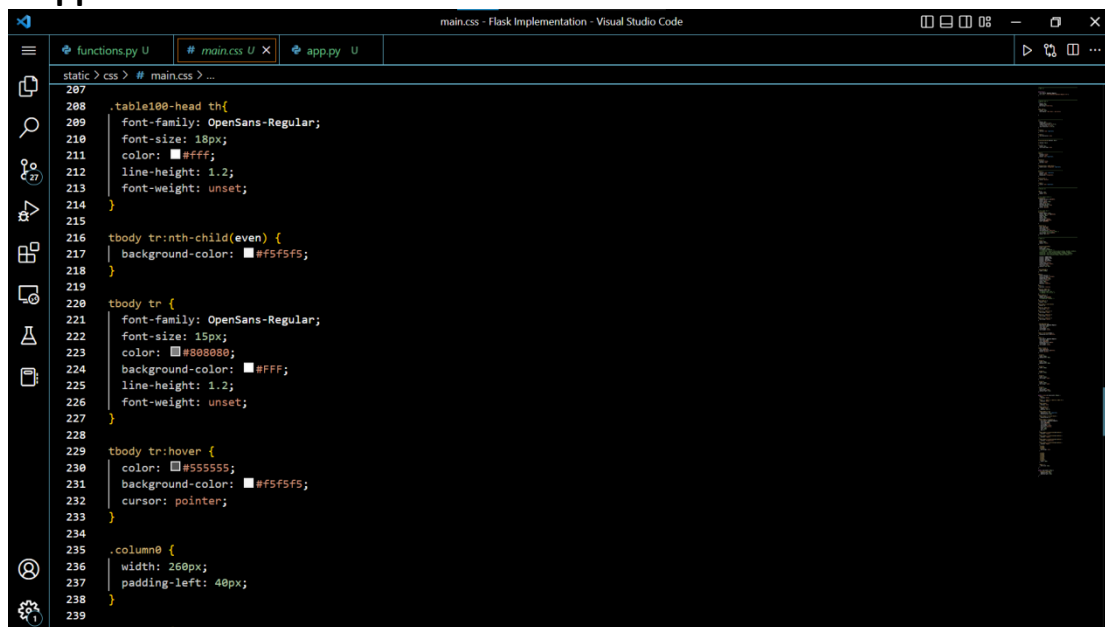
### **Integrating the Model with our website/Interface**

#### **Snippet of the HTML Code**



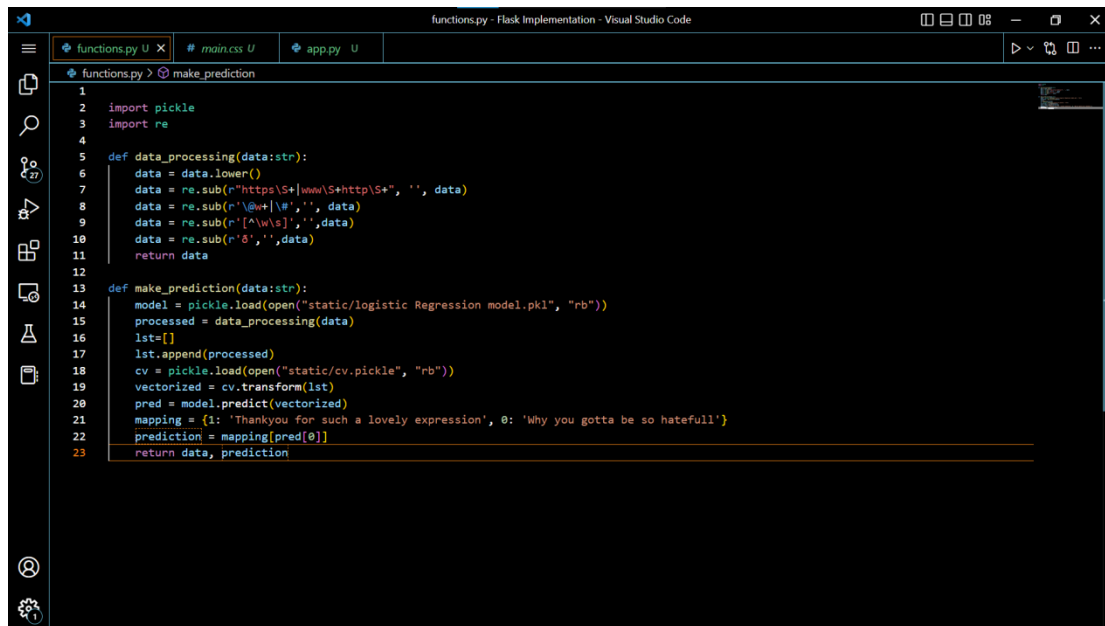
```
58
59 <form action="/">
60   <div style="text-align: center;" id="tweet-form">
61     <br>
62     <h1>Hate Speech Detector</h1>
63     <br>
64     <br>
65     <br>
66     <h2>Let's Check if the Entered text is classified as hate or not</h2>
67     <br>
68     <br>
69     <br>
70     <center>
71       <input align = "center" type="text" name="chat_in" autofocus="autofocus" value="{{x_input}}" maxlength="500" id="rcorners2" place
72     </center>
73   <br>
74   <!-- Submit button -->
75   <center>
76     <input align="center" type="submit" value="Predict" method="get" id="rcorners1">
77   </center>
78   <br>
79 </form>
80 </div>
81
82 </p>
83
84 <table>
85   <thead>
86     <tr class="table100-head">
87       <th class="column1">Text Input</th>
88       <th colspan="2" class="column1">Prediction and Hate Probability</th>
89     </tr>
90
```

## Snippet of the CSS Code



```
static > css > # main.css > ...
207
208 .table100-head th{
209   font-family: OpenSans-Regular;
210   font-size: 18px;
211   color: #fff;
212   line-height: 1.2;
213   font-weight: unset;
214 }
215
216 tbody tr:nth-child(even) {
217   background-color: #f5f5f5;
218 }
219
220 tbody tr {
221   font-family: OpenSans-Regular;
222   font-size: 15px;
223   color: #000000;
224   background-color: #fff;
225   line-height: 1.2;
226   font-weight: unset;
227 }
228
229 tbody tr:hover {
230   color: #555555;
231   background-color: #f5f5f5;
232   cursor: pointer;
233 }
234
235 .column0 {
236   width: 260px;
237   padding-left: 40px;
238 }
239
240 .column1 {
```

## Snippet of the Python Code



```
1
2 import pickle
3 import re
4
5 def data_processing(data:str):
6     data = data.lower()
7     data = re.sub(r"https\S+|www\S+http\S+", '', data)
8     data = re.sub(r'\@+|#+', '', data)
9     data = re.sub(r'\^|w|s', '', data)
10    data = re.sub(r'$', '', data)
11    return data
12
13 def make_prediction(data:str):
14    model = pickle.load(open("static/logistic Regression model.pkl", "rb"))
15    processed = data_processing(data)
16    lst=[]
17    lst.append(processed)
18    cv = pickle.load(open("static/cv.pickle", "rb"))
19    vectorized = cv.transform(lst)
20    pred = model.predict(vectorized)
21    mapping = {1: 'Thankyou for such a lovely expression', 0: 'Why you gotta be so hatefull'}
22    prediction = mapping[pred[0]]
23    return data, prediction
```

## Snippet of the Python Code



```
1 import flask
2 from flask import request
3 from functions import *
4
5 app = flask.Flask(__name__)
6
7 @app.route("/", methods=["GET"])
8 def predict():
9
10    print(request.args)
11    if(request.args):
12        x_input, pred_class, pred_proba = make_prediction(request.args['chat_in'])
13        print(x_input)
14        return flask.render_template('predictor.html',
15                                    chat_in=x_input,
16                                    prediction_class=pred_class,
17                                    )
18    else:
19        return flask.render_template('predictor.html',
20                                    chat_in=" ",
21                                    prediction_class=" ",
22                                    )
23
24
25
26 if __name__ == "__main__":
27     app.run(debug=True)
28
```

## Interface

## Hate Speech Detector

Let's Check if the Entered text is classified as hate or not

Predict

Text Input	Prediction and Hate Probability
I Hate You	Why you gotta be so hatefull

## Hate Speech Detector

Let's Check if the Entered text is classified as hate or not

Predict

Text Input	Prediction and Hate Probability
you are awesome	Thankyou for such a lovely expression

## 6. Reference

[https://drive.google.com/drive/folders/1uQiyJ\\_mDlOCcecMw7C-JYUs9bGnVJ\\_j8](https://drive.google.com/drive/folders/1uQiyJ_mDlOCcecMw7C-JYUs9bGnVJ_j8)