2019

Identifying and Categorizing Offensive Language in Social Media

Artificial Intelligence

Shorouk Shaheen 3693
Engy Ibrahim 3958
Ahmed ElKalaf 3900
Merna Adel 4168

IDENTIFYING AND CATEGORIZING OFFENSIVE LANGUAGE IN SOCIAL MEDIA

INTRO

- Social media is a very popular way for people to express their opinions publicly and to interact with others online. In aggregation, social media can provide a reflection of public sentiment on various events. Unfortunately, many users engaging online, either on social media, forums or blogs, will often have the risk of being targeted or harassed via abusive language, which may severely impact their online experience and the community in general. The existence of social networking services creates the need for detecting user-generated hateful messages prior to publication.

All published text that is used to express hatred towards some particular group with the intention to humiliate its members is considered a hateful message.

PROBLEM STATEMENT

➔ Twitter has more or less than 310 million monthly active users with total 500 million of tweets per day. Tweet is a 140-characters message that can contain opinion or information; tweets are not properly structured because users do not care about spelling and grammatical construction when posting their tweets.

➔ **#Steps of Classification:**

➔ **A. Fetching Tweets :**

Fetching tweets is a process of getting the tweets directly from Twitter server using Twitter API (Application Programming Interface) based on keywords used.

➔ **B. Preprocessing :**

Preprocessing is a process of cleaning noise like links, punctuation or stop words that does not contain any useful information in the text. There are few steps of preprocessing used in this research such as :

- Remove URLs, to remove unwanted URL like http, https or something like them in the text.

-Remove punctuation, to remove punctuation marks like Tokenizing, to break text into each word.

-Remove stop words, to remove words like a, most, and, is and so on in the text because those words do not contain any useful information.

-Stemming, to change a word in the text into its base term or root term. Example, happiness to happy.

➔ **C. Text Feature Extraction :**

Text Feature Extraction is a process of converting text into set of features in real number form side a vector that will be used as input for classification. The process of extracting feature from text in this research used Bag of Words model that transforms all texts into a dictionary consist of all words appear in all texts. It later creates set of features in real number form inside a vector for each text where the value of each feature inside vector will be based on the frequency of each word counted in the text.

➔ **D. Machine Learning :**

Machine Learning is one of computer science branches that focuses on providing the technology the ability to learn and adapt from given data so technology can learn and grow independently without being programmed explicitly by developers.

➔ **Sub-task A - Offensive language identification:**

To know whether the input data file contain offensive words or not.

NOT= Posts don’t contain offensive language

OFF= Posts that contain offensive language

APPROACH

Data set

➔ **Training set:**

we get it from ‘training-v1’ from codelab

➔ **Test set:**

we get it from ‘trial-data’ from codelab

we read the data set using csv library and split the columns by excel-tab and then we get the column number 1 tweets and column number 2 sub-task A

➔ **Notes about training set:**

Total number of tweets = 13240 tweet

number of OFF tweets = 4400 tweet

number of not OFF tweets = 8840 tweet (which is nearly double the OFF which means that data is unbalanced)

➔ **Notes about test set:**

Total number of tweets = 319 tweet

number of OFF tweets = 77 tweet

number of not OFF tweets = 242 tweet (which is about trible the OFF which means that data is unbalanced)

Vectorizer

We used Tf-idf with four parameters:

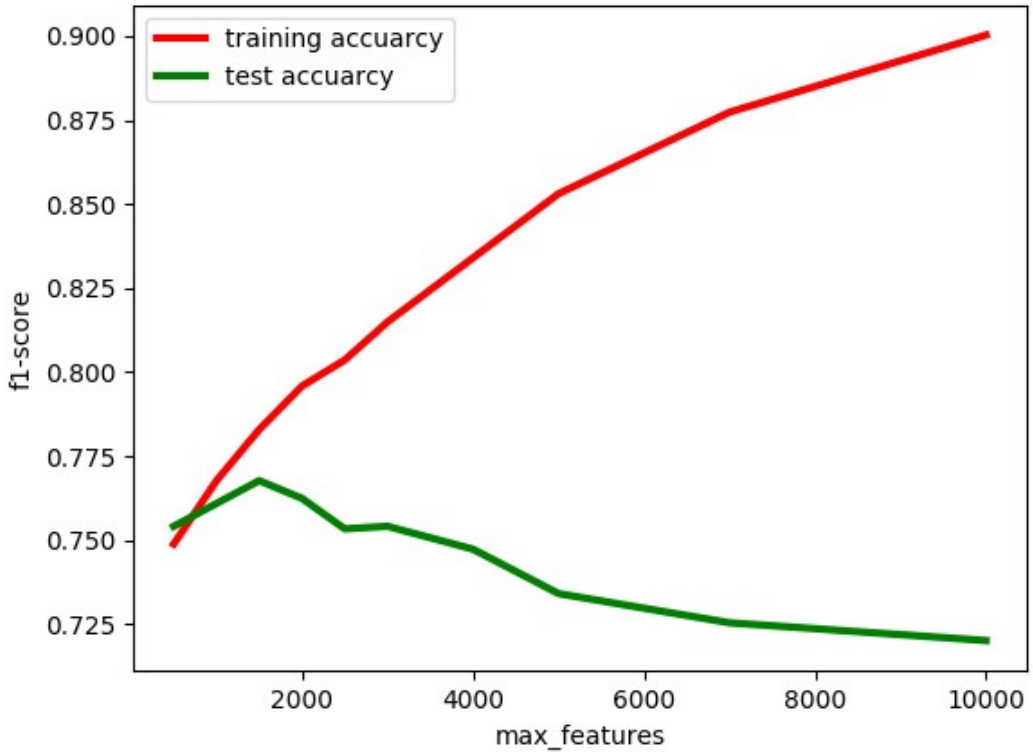
1) max_features:

we tried this values

[500,1000,1500,2000,2500,3000,4000,5000,7000,10000]

and made the classification using Naive bayes classifier as it is the fastest one .

And this graph represents the results



➔ **Observation:**

max cross validation set f1-score 0.7677492447129909 for max_features = 1500

max training set f1-score 0.9003021148036254 for max_features = 10000 ,but

Cross validation set f1-score = 0.7201661631419939

➔ **Conclusion**

Too much features lead to overfitting

2) min_df:

we choose it's value = 5 to make the word as feature if only it appears in more than 5 documents , to not the rare words and improve accuarcy.

3) max_df:

we choose it’s value = 0.7 to make the word as feature if only it appears in less than 70% of documents. To remove the words that appears at any type of documents and will not affect classification

4) stop_words:

we select stop words = ‘english’ to remove the most common english words that will not help in classification like (the,is,are...etc.)

PRE-PROCESSING

➔ **Introduction**

It is easy to forget how much data is stored in the conversations we have every day. With the evolution of the digital landscape, tapping into text, or Natural Language Processing (NLP), is a growing field in artificial intelligence and machine learning. This article covers the common pre-processing concepts applied to NLP problems.

Text can come in a variety of forms from a list of individual words, to sentences to multiple paragraphs with special characters (like tweets for example). Like

any data science problem, understand the questions that are being asked will inform what steps may be employed to transform words into numerical features that work with machine learning algorithms.

➔ **The Importance of Pre-Processing**

Transforming text into something an algorithm can digest it a complicated process. There are four different parts:

- Cleaning consist of getting rid of the less useful parts of text through stopword removal, dealing with capitalization and characters and other details.
- Annotation consists of the application of a scheme to texts. Annotations may include structural markup and part-of-speech tagging.
- Normalization consists of the translation (mapping) of terms in the scheme or linguistic reductions through Stemming, Lemmazation and other forms of standardization.
- Analysis consists of statistically probing, manipulating and generalizing from the dataset for feature analysis.

➔ **The Tools**

There are a variety of pre-processing methods. The list below is far from exclusive but it does give an idea of where to start. It is important to realize, like with all data problems, converting anything into a format for machine learning reduces it to a generalized state which means losing some of the fidelity of the data along the way. The true art is understand the pros and cons to each to carefully chose the right methods.

1]Capitalization

Text often has a variety of capitalization reflecting the beginning of sentences, proper nouns emphasis. The most common approach is to reduce everything to lower case for simplicity but it is important to remember that some words, like “US” to “us”, can change meanings when reduced to the lower case.

2]Stopword

A majority of the words in a given text are connecting parts of a sentence rather than showing subjects, objects or intent. Word like “the” or “and” cab be removed by comparing text to a list of stopword.

IN:

['He', 'did', 'not', 'try', 'to', 'navigate', 'after', 'the', 'first', 'bold', 'flight', ',', 'for', 'the', 'reaction', 'had', 'taken', 'something', 'out', 'of', 'his', 'soul', '.']

OUT:

['try', 'navigate', 'first', 'bold', 'flight', ',', 'reaction', 'taken', 'something', 'soul', '.']

In the example above it reduced the list of 23 words to 11, but it is important to note that the word “not” was dropped which depending on what I am working on could be a large problem. One might created their own stopword dictionary manually or utilize prebuilt libraries depending on the sensitivity required.

3]Tokenization

Tokenization describes splitting paragraphs into sentences, or sentences into individual words. For the former Sentence

Boundary Disambiguation (SBD) can be applied to create a list of individual sentences. Sentences can be split into individual words and punctuation through a similar process. Most commonly this split across white spaces, for example:

IN:

"He did not try to navigate after the first bold flight, for the reaction had taken something out of his soul."

OUT:

['He', 'did', 'not', 'try', 'to', 'navigate', 'after', 'the', 'first', 'bold', 'flight', ',', 'for', 'the', 'reaction', 'had', 'taken', 'something', 'out', 'of', 'his', 'soul', '.']

4]Stemming

Much of natural language machine learning is about sentiment of the text. Stemming is a process where words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix. There are several stemming models, including Porter and Snowball. The results can be used to identify relationships and commonalities across large datasets.

IN:

["It never once occurred to me that the fumbling might be a mere mistake."]

OUT:

['it', 'never', 'onc', 'occur', 'to', 'me', 'that', 'the', 'fumbl', 'might', 'be', 'a', 'mere', 'mistake.'],

It is easy to see where reductions may produce a “root” word that isn’t an actual word. This doesn’t necessarily adversely affect its efficiency, but there is a danger of

“overstemming” were words like “universe” and “university” are reduced to the same root of “univers”.

5] convert emojis to their names

so later processing stages were easier to do.

6] remove hyperlinks

➔ **Conclusion**

While this is far from a comprehensive list, preparing text is a complicated art which requires choosing the optimal tools given both the data and the question you are asking. Many pre-built libraries and services are there to help but some may require manually mapping terms and words.

Once a dataset is ready supervised and unsupervised machine learning techniques can be applied. From my initial experiments, which will be it’s own article, there is a sharp difference in applying preprocessing techniques on a single string compared to large dataframes. Tuning the steps for optimal efficiency will be key to remain flexible in the face of scaling.

#Classifiers:

1- Logistic Regression:

➔ Logistic Regression is one of many machine learning methods that works by taking input and multiplied the

input value with weight value. It is a classifier that learns what features from the input that are the most useful to discriminate between the different possible classes.

****Parameters****

***penalty** *

*Used to specify the norm used in the penalization. New in version 0.19: l1 penalty with SAGA solver (allowing ‘multinomial’ + L1)

***dual** *

*Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver.

C : float

*Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

***fit_intercept** *

*Specifies if a constant . should be added to the decision function.

***intercept_scaling** *

*Useful only when the solver ‘liblinear’ is used and self.fit_intercept is set to True. In this case, x becomes [x, self.intercept_scaling], i.e. a “synthetic” feature with constant value equal to intercept_scaling is appended to the instance vector. The intercept

***class_weight** *

*Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.

*Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

***random_state :** The seed of the pseudo random number generator to use when shuffling the data. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator;

****solver : .**

****max_iter :** int

****multi_class :**

*If the option chosen is ‘ovr’, then a binary problem is fit for each label. For ‘multinomial’ the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. ‘multinomial’ is unavailable when solver=’liblinear’. ‘auto’ selects ‘ovr’ if the data is binary, or if solver=’liblinear’, and otherwise selects ‘multinomial’.

****verbose :** int, default: 0

*For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.

****warm_start**

*When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

** **n_jobs :** int or None, optional (default=None)

*Number of CPU cores used when parallelizing over classes if multi_class=’ovr’”. This parameter is ignored when the solver is set to ‘liblinear’ regardless of whether ‘multi_class’ is specified or not. None means 1 unless in Algorithm to use in the optimization problem.

• For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones.

• For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.

•‘newton-cg’, ‘lbfgs’ and ‘sag’ only handle L2 penalty, whereas ‘liblinear’ and ‘saga’ handle L1 penalty.

****max_iter :** int

2- Naïve Bayes

➔ The Naive Bayes algorithm is well known in machine learning. The algorithm uses Bayes' Theorem to calculate the probability of an attribute belonging to a particular class. It is called a "naive" because it assumes that the attributes are independent, a naive premise.

➔ In the text document classification, each attribute to be classified are the words in the document. Although the Bayesian classifier is supported by statistical theories it is necessary to make sure that the attributes are independent of each other before using naive Bayesian. Bayesian classifiers can handle both nominal and numeric attributes well for nominal prediction. The Naive Bayes classifier has two different models. The multinomial model seems to be more realistic for our problem. Because a word considered "dangerous" may appear in a "normal" conversation sometimes, but it will be more frequent in dangerous conversations.

****Naive Bayes classifier for multinomial models****

➔ The basic idea of Naive Bayes technique is to find the probabilities of classes assigned to texts by using the joint probabilities of words and classes.

****Parameters****

***alpha:** float, optional (default=1.0)

*Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

***fit_prior:** boolean, optional (default=True)

*Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

***class_prior:** array-like, size (n_classes,), optional (default=None)

*Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

3- Support Vector Machine

➔ A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

****Parameters****

****Kernel****

*The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.

*For linear kernel the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

* f(x) = B(0) + sum(ai * (x,xi))

*This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

The polynomial kernel can be written as K(x,xi) = 1 + sum(x * xi)^d and exponential as K(x,xi) = exp(-gamma * sum((x — xi^2)).

****Regularization****

*The Regularization parameter (often termed as C parameter in python’s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.

*For large values of C, the optimization will choose a smaller-margin hyper plane if that hyper plane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyper plane misclassifies more points.

****Gamma****

*The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation.

****Margin****

*A margin is a separation of line to the closest class points.

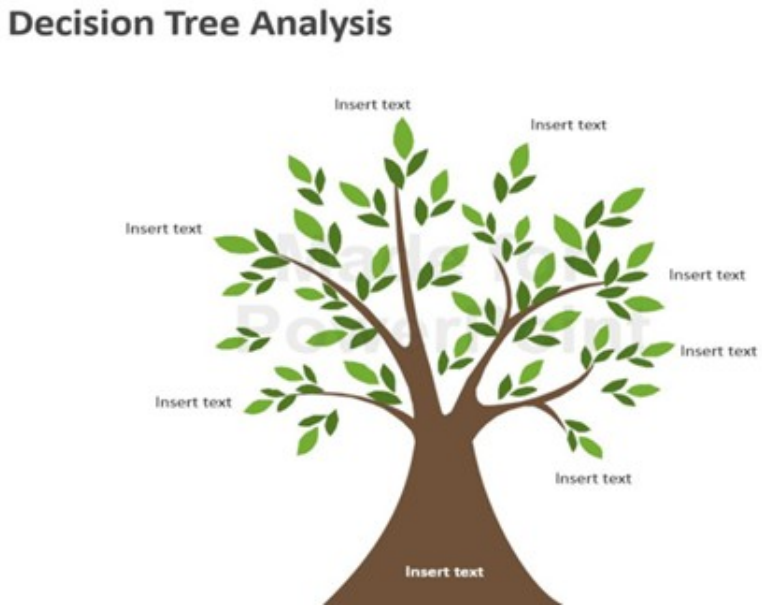
*A good margin is one where this separation is larger for both the classes, allows the points to be in their respective classes without crossing to other class.

4- Decision Tree

➔ Decision trees are supervised learning algorithms used for both, classification and regression tasks where we will concentrate on classification in this first part of our decision tree tutorial.

➔ Decision trees are assigned to the information based learning algorithms which use different measures of information gain for learning. We can use decision trees for issues where we have continuous but also categorical input and target features. The main idea of decision trees is to find those descriptive features which contain the most "information" regarding the target feature and then split the dataset along the values of these features such that the target feature values for the resulting sub_datasets are as pure as possible --> The descriptive feature which leaves the target feature most purely is said to be the most informative one. This process of finding the "most informative" feature is done until we accomplish a stopping criteria where we then finally end up in so called leaf nodes. The leaf nodes contain the predictions we will make for new query instances presented to our trained model. This is

possible since the model has kind of learned the underlying structure of the training data and hence can, given some assumptions, make predictions about the target feature value (class) of unseen query instances.



1. using scaling features

**The result of standardization (or Z-score normalization) is that the features will be rescaled between 0 ,1

2.using standard scaler

**Standardize features by removing the mean and scaling to unit variance

*The standard score of a sample x is calculated as:

z = (x - u) / s

3.Principal component analysis (PCA)

**Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

**then use Feature Selection Techniques:

*In machine learning, Feature selection is the process of choosing variables that are useful in predicting the response (Y). It is considered a good practice to identify which features are important when building predictive models.

****Best****

1-criterian=gini

2-min samples leaf=30

3-min sample split=15

4-max depth=15

5-splitter →best

6-randomstate →none

7-max_leaf_nodes default=None

****conclusion****

*before tuning parameter we use scaling features and Feature Selection accuracy increase but over fitting exist

*when we tuning parameter over fitting(accuracy of training data) decrease ,accuracy to validation increase and test data .

RESULTS

We have 2,648 testing instance out of 13,240 instances, As the test data are 20% of the original data.

- ➔ **Precision or Specificity:**
  How many selected instances are relevant.
- ➔ **Recall or Sensitivity:**
  How many relevant instances are selected.
- ➔ **F1 score:**
  Harmonic mean of precision and recall.

Logistic Regression

Training

Confusion matrix

```
[[6619  415]
 [1663 1895]]
Classification report
              precision    recall  f1-score   support

         NOT       0.80      0.94      0.86      7034
         OFF       0.82      0.53      0.65      3558

   micro avg       0.80      0.80      0.80     10592
   macro avg       0.81      0.74      0.76     10592
weighted avg       0.81      0.80      0.79     10592

f1 score: 0.8038141993957704
Cross validation
Confusion matrix
[[1676  130]
 [ 439  403]]
Classification report
              precision    recall  f1-score   support

         NOT       0.79      0.93      0.85      1806
         OFF       0.76      0.48      0.59       842

    micro avg       0.79      0.79      0.79      2648
    macro avg       0.77      0.70      0.72      2648
 weighted avg       0.78      0.79      0.77      2648

f1 score: 0.7851208459214503
Test
Confusion matrix
[[231  11]
 [ 29  48]]
Classification report
              precision    recall  f1-score   support
```

```
           NOT       0.89      0.95      0.92       242
           OFF       0.81      0.62      0.71        77
    micro avg       0.87      0.87      0.87       319
    macro avg       0.85      0.79      0.81       319
 weighted avg       0.87      0.87      0.87       319
Accuarcy: 0.8746081504702194
```

Naive Bayes

```
Training
Confusion matrix
[[6876  158]
 [2310 1248]]
Classification report
              precision    recall  f1-score   support
         NOT       0.75      0.98      0.85      7034
         OFF       0.89      0.35      0.50      3558
    micro avg       0.77      0.77      0.77     10592
    macro avg       0.82      0.66      0.68     10592
 weighted avg       0.80      0.77      0.73     10592
f1 score: 0.7669939577039275
Cross validation
```

Confusion matrix

[[1758 48]

[591 251]]

Classification report

	precision	recall	f1-score	support
NOT	0.75	0.97	0.85	1806
OFF	0.84	0.30	0.44	842
micro avg	0.76	0.76	0.76	2648
macro avg	0.79	0.64	0.64	2648
weighted avg	0.78	0.76	0.72	2648

f1 score: 0.7586858006042296

Test

Confusion matrix

[[238 4]

[37 40]]

Classification report

	precision	recall	f1-score	support
NOT	0.87	0.98	0.92	242
OFF	0.91	0.52	0.66	77
micro avg	0.87	0.87	0.87	319
macro avg	0.89	0.75	0.79	319
weighted avg	0.88	0.87	0.86	319

Accuarcy: 0.8714733542319749

Decision tree

Training

Confusion matrix

[[6800 234]

 [2370 1188]]

Classification report

	precision	recall	f1-score	support
NOT	0.74	0.97	0.84	7034
OFF	0.84	0.33	0.48	3558
micro avg	0.75	0.75	0.75	10592
macro avg	0.79	0.65	0.66	10592
weighted avg	0.77	0.75	0.72	10592

f1 score: 0.754154078549849

Cross validation

Confusion matrix

```
[[1747   59]
 [ 574  268]]
Classification report
              precision    recall  f1-score   support

         NOT       0.75      0.97      0.85      1806
         OFF       0.82      0.32      0.46       842

    micro avg       0.76      0.76      0.76      2648
    macro avg       0.79      0.64      0.65      2648
 weighted avg       0.77      0.76      0.72      2648

f1 score: 0.7609516616314199
Test
Confusion matrix
[[229  13]
 [ 33  44]]
Classification report
              precision    recall  f1-score   support

         NOT       0.87      0.95      0.91       242
         OFF       0.77      0.57      0.66        77

    micro avg       0.86      0.86      0.86       319
    macro avg       0.82      0.76      0.78       319
 weighted avg       0.85      0.86      0.85       319

Accuarcy: 0.8557993730407524
```

SVM

```
Training
Confusion matrix
[[6636  398]
 [1707 1851]]
Classification report
              precision    recall  f1-score   support

         NOT       0.80      0.94      0.86      7034
         OFF       0.82      0.52      0.64      3558

   micro avg       0.80      0.80      0.80     10592
   macro avg       0.81      0.73      0.75     10592
weighted avg       0.80      0.80      0.79     10592

f1 score: 0.8012651057401813
Cross validation
Confusion matrix
[[1685  121]
 [ 456  386]]
```

Classification report

	precision	recall	f1-score	support
NOT	0.79	0.93	0.85	1806
OFF	0.76	0.46	0.57	842
micro avg	0.78	0.78	0.78	2648
macro avg	0.77	0.70	0.71	2648
weighted avg	0.78	0.78	0.76	2648

f1 score: 0.7820996978851964

Test

Confusion matrix

[[231 11]

[27 50]]

Classification report

	precision	recall	f1-score	support
NOT	0.90	0.95	0.92	242
OFF	0.82	0.65	0.72	77
micro avg	0.88	0.88	0.88	319
macro avg	0.86	0.80	0.82	319
weighted avg	0.88	0.88	0.88	319

Accuarcy: 0.8808777429467085

DISCUSSION

➔ By trying all of the 4 algorithms, we have noticed that SVM got the best Accuracy which is approximated to 88%