SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media

Asmaa Salama, Menna Ghanem, Guehad Mohamed, Esraa Elhawash
Faculty of Engineering, Alexandria University, Alexandria
Department of Computer and Communications

Abstract

This paper describes task 6 of SemEval, "Identifying and Categorizing Offensive Language in Social Media", which includes identifying offensive and aggressive expressions in different platforms of social media.

1 Introduction

The identification of offensive language in social media is an important field of study. The task usually involves detecting whether a piece of text expresses a negative approach towards people, races or products. We performed a classification of different texts in a way that determines whether it's offensive or not offensive. However, we can't identify if the offensive text is targeted or not. We used some machine learning classifiers specified for text analysis. Some if which are:

1. **Naive Bayes:** calculates the probabilities for every factor. Then it selects the outcome with highest probability. This classifier assumes the features are independent. Hence, the word naive.

2. **Logistic Regression:** Logistic regression is a statistical method for analyzing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

3. **K-Nearest Neighbor (KNN):** K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.

4. **Support Vector Machine (SVM):** is a discriminating classifier formally defined by a separating hyper-plane.

5. **Decision tree:** In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

6. **Random Forest:** A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

2 Problem statement

Offensive language is pervasive in social media. Individuals frequently take advantage of the perceived anonymity of computer-mediated communication, using this to engage in behavior that many of them would not consider in real life. Online communities, social media platforms, and technology companies have been investing heavily in ways to cope with offensive language to prevent abusive behavior in social media.

One of the most effective strategies for tackling this problem is to use computational methods to identify offense, aggression, and hate speech in user-generated content (e.g. posts, comments, microblogs, etc.).

In OffensEval we break down offensive content into three sub-tasks taking the type and target of offenses into account.

Sub-tasks:

1. **Sub-task A** Offensive language identification.

2. **Sub-task B** Automatic categorization of offense types.

3. **Sub-task C** Offense target identification.

We worked mainly on sub-task A, which includes classifying the text into 2 separate classes. One of which is: "Offensive",and the other is: "NOT Offensive".

We were given a training data set, which includes thousands of entries in the form of tweets. Each entry was labeled, "OFF" or, "NOT" to in order to indicate the offensiveness of words in the text.

3 Approach

The Training Data set is a collection of enriched labeled data to help us train our models, the more data we have, the sharper the model accuracy. A training data set was provided to us by the SemEval-2019 website, its all in English , all of them are tweets in random topics from random people.

The data was in the form of columns, each entry has a unique id written before it, a username handle written in the form @USER followed by the actual tweet and then in the last column lies the label of that specific tweet. The first step we did was the pre-processing step.

3.1 Pre-Processing

The first thing we did was opening the .tvs file and then we tokenized the entries of the file splitting on the tab spaces since the file was in the form of columns. Then we got rid of the username handle from all the entries since it wont have any effect on the classifying. Next we took the text column content and put it in an array, and the labels column which has the labels of each feature and put them in a separate array.

We used the TFIDF count vectorizer to do so we had to import the first library we used, Sci-kit Learn , this has a built in function stop= english what this does is remove the unnecessary redundant English words from each entry in the text array which holds the tweets for example it turns the sentence i am in a very bad mood today it returns the following bad mood the only two words that could affect the meaning of the sentence in a way that would help my classification. This concludes the pre-processing step, the output of this is then classified using our 4 classifiers.

3.2 Vectorization

Although a short process, its an extremely important step because all classifiers work on vectors not actual strings. We then use the fit transform function that takes the array of texts and transforms it to a vector representing the words that correspond in a particular array with the entire corpus. The same idea is then done for the labels array because entries need to be transformed to integers , so for every OFF label its represented with a 1 and every NOT label is represented with a 0. now our training data is ready for the classification phase.

3.3 Classifiers

1. **Naive Bayes** Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting. It can also be represented using a very simple Bayesian network. Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection. Seeing how famous Naive Bayes has been used in text classification we decided to start with it in our implementation.

   Using the same Sci-kit library we imported MultinomialNB which is suitable for classification with discrete features. We use the fit function in this classifier taking as parameters the array of training texts and the training labels, both of which are vectors of integer value. After this step we use the function predict() on a temporary test data set file and we print the results of the predictions returned in an array and the result classifies each tweet according to the fitted data from the training set.

2. **Logistic Regression** Logistic Regression is a go-to method for binary classification (which is what were going for), This algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. We need the output of the algorithm to be class variable, i.e 0-no, 1-yes. Therefore, we are squashing the output of the linear equation into a range of [0,1]. To squash the predicted value between 0 and 1,

we use the sigmoid function(the logistic regression). Its widely used for text classification and is known for exhibiting pretty accurate result , also its very simple to implement.

We import the Logistic Regression library again from the Scit-kit library and we use the exact functions used previously in Naive Bayes, fit() and predict() taking in the training data and the test data respectively and resulting in predictions in the form of an array.

3. **K-Nearest Neighbor (KNN)** The KNN classifier is based on the assumption that the classification of an instance is most similar to the classification of other instances that are nearby in the vector space. Compared to other text categorization methods such as Bayesian classifier, KNN does not rely on prior probabilities, and it is computationally efficient.

   KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry, this algorithm is one of the simplest classification algorithm it also has a low calculation time. The challenge is to find the perfect k to run our algorithm with trying to avoid underfitting or overfitting. KNN algorithm is used to classify by finding the K nearest matches in training data and then using the label of closest matches to predict.

   We import the KNeighborsClassifier from the Sci-kit library and we use the same functions previously explained in the other classifiers with only the difference of setting the k to the desired value before starting the fitting operation, the result as the other classifiers is an array.

4. **Support Vector Machine (SVM)** Support vector machines is an algorithm that determines the best decision boundary between vectors that belong to a given group (or category) and vectors that do not belong to it. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.

   SVMs can produce accurate, unique and robust classification results on a sound theoretical basis, even when input data are non-monotone and non-linearly separable. So they can help to evaluate more relevant information in a convenient way. Since they linearize data on an implicit basis by means of kernel transformation, the accuracy of results does not rely on the quality of human expertise judgment for the optimal choice of the linearization function of non-linear input data.

   Svm library is imported from Sci-kit library and we use the function SVC(support vector classifier) and we set the kernel parameter to linear. And we fit the data and predict them as mentioned before in the other classifiers.

5. **Decision Tree** Decision trees implicitly perform variable screening or feature selection.It Can handle both numerical and categorical data, also the effort required for Data preparation is minimal.

   we imported DecisionTreeClassifier from the Sci-kit library to use for fitting our model and predicting labels for the test data. we chose gini as our criterion because it's better and faster then entropy at calculating the gain that decides where the tree splits.

6. **Random Forest** Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. They are adaptable at solving any kind of problem at hand. Random Forest is a versatile machine learning method capable of performing both regression and classification tasks.

   It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model. One of benefits of Random forest is the power of handle large data set with higher dimensionality.

   It can handle thousands of input variables and identify most significant variables so it

is considered as one of the dimensionality reduction methods. We imported the RandomForestClassifier from Sci-kit ensemble library and we use it to fit and predict our model.

4 Results

We ran our code on a test file full of thousands of tweets provided to us by the SemEval-2019, we used each classifier on the file and we calculated the accuracy of all of them and the results were as following:

Classifier	Accuracy	Confusion Matrix
NB	72.6%	[1564 525] [200 359]
LR	74.9%	[1531 431] [233 453]
KNN	67.8%	[1684 771] [80 113]
SVM	76.6%	[1608 464] [156 420]
DT	70.9%	[1393 397] [371 487]
RF	76.3%	[1644 512] [120 372]

Each classifier results in an output in the form of a vector of zero's and one's e.g([0101011100000010001111000.....001]).
1 to indicate an entry being offensive and 0 indicating it's not. We made some adjustments in a way that transforms the vector back into text, so now the code's output is each text in the test file written next to it 'NOT' for a classification of a non offensive tweet and 'OFF' if the tweet is in fact offensive in anyway.

5 Discussion

5.1 Evaluation

In order to evaluate the performance of our classifiers individually we had a function that calculates the accuracy for each classifier. We split the training data with the help of sklearn library and we took a percentage of the entire training data set file to count as the training data which we fit on and the other part of the file is used for validation. Each part of the Training data file is split into texts and labels to use for both the training and the validation and they both go through the pre-processing phase in order to get accurate results. The accuracy is calculated simply by fitting the part of training data designated for the validation by looping around the labels and counting the ones that were predicted correctly and after all of them are counted it's divided by the length of all predictions and that represents the accuracy of our classifier. We also used a confusion matrix which is a breakdown of predictions into a table showing correct predictions (the diagonal) and the types of incorrect predictions made (what classes incorrect predictions were assigned). for a better indication of the precision (exactness) of our classification.

5.2 Tuning parameters

We needed to improve accuracy of the results from our classifiers and one way to do that is tuning the parameters of each classifier in a way the maximizes the performance of classification. Some of the parameters we changed from their default state were:

1. **Naive Bayes** There are 3 parameters: alpha, fit prior, class prior. Alpha: a smoothing parameter (int) to increase the probabilities in our non-maximum likelihood equation by 1 to make everything non-zero. fit prior: whether to learn class prior probabilities or not.

2. **Logistic Regression** penalty parameter used to regularize the model. It basically adds the penalty as model complexity increases to avoid over fitting. We have used L2 which is called Ridge Regression. With testing we set the c parameter to 10 which has provided the best accuracy and increased the number of iterations to 1000 after which the model converges

3. **K-Nearest Neighbor (KNN)** was tuned using a code implemented to find the best K to run the algorithm with.

4. **Support Vector Machine (SVM)** we tuned some important parameters kernel and C. we set the kernel to: linear because a large number of features is more likely that the data is linearly separable, C: Penalty parameter C of the error term, it controls the trade off between smooth decision boundary and classifying the training points correctly setting it to 1 proved the best results

5.3 Conclusion

In our implementation what we gathered is that the best classifier was the SVM and it worked with accuracy 76.3% This is due to the fact that the SVM works really well with clear margin of separation and it's heavily recommended for text classification projects.