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

Zeyad El-Zanaty |3320 Omar Aly |3405 Kareem Hossam |3317

Abstract

The objective of this paper is to provide a description for a classification system built for SemEval-2019 Task 6: OffensEval. This system classifies a tweet as either offensive or not offensive (Sub-task A) and further classifies offensive tweets into categories (Sub-tasks B – C). Some sort of grid search approach is taken where multiple techniques for preprocessing, feature extraction and classification are implemented and combinations of them all are tried to achieve the best model for the given dataset.

1 Introduction

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.). This topic has attracted significant attention in recent years as evidenced in recent publications (Waseem et al. 2017; Davidson et al., 2017, Malmasi and Zampieri, 2018, Kumar et al. 2018)

1.1 Problem Statement

In OffensEval we break down offensive content into three sub-tasks taking the **type** and **target** of offenses into account.
Sub-task A - Offensive language identification;
In this sub-task we are interested in the identification of offensive posts and posts containing any form of (untargeted) profanity. In this sub-task there are 2 categories in which the tweet could be classified –
Not Offensive - This post does not contain offense or profanity. Non-offensive posts do not include any form of offense or profanity.
Offensive - This post contains offensive language or a targeted (veiled or direct) offense.
To sum up this category includes insults, threats, and posts containing profane language and swear words.

Sub-task B - Automatic categorization of offense types;
In this sub-task we are interested in categorizing offenses. Tweets are labeled from one of the following categories –
Targeted Insult - A post containing an insult or a threat to an individual, group, or others;
Untargeted - A post containing non-targeted profanity and swearing.
Posts containing general profanity are not targeted but they contain non-acceptable language. On the other hand, insults and threats are targeted at an individual or group.

Sub-task C: Offense target identification.
Finally, in sub-task C we are interested in the target of offenses. Only posts which are either insults or threats are included in this sub-task. The three categories included in sub-task C are the following:
Individual - The target of the offensive post is an individual: a famous person, named individual or an unnamed person interacting in the conversation.
Group - The target of the offensive post is a group of people considered as a unity due to the same ethnicity, gender or sexual orientation, political affiliation, religious belief, or something else.
Other – The target of the offensive post does not belong to any of the previous two categories (e.g. an organization, a situation, an event, or an issue).

1.2 Dataset

The dataset used is the one provided by SemEval, it contains 13,240 annotated tweets. It was annotated using crowdsourcing. The gold labels were assigned taking the agreement of three annotators into consideration. No correction has been carried out on the crowdsourcing annotations.

2 Implementation

The system is a combination of three essential layers. **First**, preprocessing which is a necessary step in NLP as textual data could and most likely will not be clean, thus will affect further stages such as classification and create an incoherent model. **Second**, feature extraction or vectorization, which translates words to a number or (series of numbers) with different weights to represent this word. **Finally**, classification is where the magic happens. Features extracted from the previous step is fed into a learner and a model is created for further testing. We will dive into these layers with more details.

2.1 Text Preprocessing

A tweet contains many unwanted data that would take extra computational power and decrease the accuracy of the results. So, noise removal and some normalization techniques must be applied to the corpus in-order to generate more consistent models.

2.1.1 Tokenization

The first step to work with textual data is tokenization, which is the process of splitting the tweet into an array of words. All the next operations work on a single word so tokenization is a must.

2.1.2 Stopwords Removal

Next, noise removal, the filtering of words that don't have significance in the context of the sentence. They are known as stopwords, without them the context of the tweet won't be affected and we grasp the same subject of that tweet. It's done by iterating over the array and removing stopwords provided by the NLTK library while also removing some punctuations and emojis.

2.1.3 Lemmatization/Stemming

There's been debate on whether to lemmatize or stem the word to get better results. They both strip the word of their grammar and convert them to their roots in the English language, but they do operate differently.
For *stemming*, PorterStemmer provided by NLTK was used which strips the words of their suffixes such as {'ing', 'ly', 'es', 's'} according to the Porter stemming algorithm by Martin Porter. For example, {'love', 'loved', 'loves' 'lovely', 'loving'} all would be stemmed to 'love'.
As for *lemmatization*, first, words are part-of-speech tagged as {noun, verb, adjective, adverb}, then converts the word to its root, also called a lemma. It makes use of morphological analysis (word structure and grammar relations). NLTK library was used for lemmatization as well.

2.2 Feature Extraction

Now that we've got our clean almost noise-free textual data, we can't simply feed a classification model a bunch of text words, most models only work with numerical data. This is where we covert words to numerical features using one the methods mentioned below to create our classification-ready dataset.

2.2.1 TFIDF/Count

TFIDF and Count are one of the simplest type of vectorization techniques.
Count first builds the vocabulary dictionary where keys are the words available in the corpus and value is the index of the word in a vector, the vector's size is the whole number of unique words in the corpus each index is a word mapped by the vocabulary dictionary and its value is the number of occurrence of this word in the corresponding sentence. Finally, we end up with our dataset of size $\{n - samples \times size\ of\ vocabulary\}$.
TFIDF, which stands for ***Term Frequency – Inverse Document Frequency*** starts just like count but doesn't just replace a word with its count, it replaces a word according to the following formula.

$$w_{i,j} = tf_{i,j} \times \log\left({}^{N}/_{df_i}\right)$$

$$tf_{i,j} = number\ of\ occurences\ of\ word\ i\ in\ scentence\ j$$
$$df_i = number\ of\ samples\ containg\ word\ i$$
$$\mathrm{N} = total\ number\ of\ samples\ (tweets)$$

Inverse Document Frequency (IDF) is defined as logarithm of ratio of total samples available in the corpus and number of samples containing a unique word. TF IDF formula gives the relative importance of a word in a corpus.

2.2.2 Word Embedding

Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words. Word embedding in basically a vector representation of a word in a corpus, there are many available models to work with, the following 3 variations are used.

i. *Word2Vec* [Trained]– It’s one of the most popular technique to learn word embeddings using shallow neural network. It was used using genism library and the model was trained on the provided dataset with embedding size of 100.
ii. *GloVe* [Pre-trained] – GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It was trained on 2 Billion tweets – 27 Billion tokens – 1.2 million vocabulary, with embedding dimension of size 100.
iii. *fastText* – It’s a library for learning of word embeddings and text classification created by Facebook's AI Research lab. The model allows to create an unsupervised learning or supervised learning algorithm for obtaining vector representations for words. It was trained on the provided dataset with embedding size of 100.

The only problem is every sample has different dimensions, which creates an inconsistent dataset. So, 0 – padding was applied to the entire samples with the maximum length vector in the samples. All three models have been trained and available for immediate loading.

2.3 Classification

This is where all the previous work comes together for the final layer of the system. A heap of supervised learners where implemented and tuned using sci-kit learn’s GridSearchCV, which does a cross validation search on a list of hyper-parameters for a given model. As well as a Deep Learning model was adapted from Georgios K. Pitsilis, Heri Ramampiaro and Helge Langseth ‘s publication ‘Detecting Offensive Language in Tweets Using Deep Learning’.

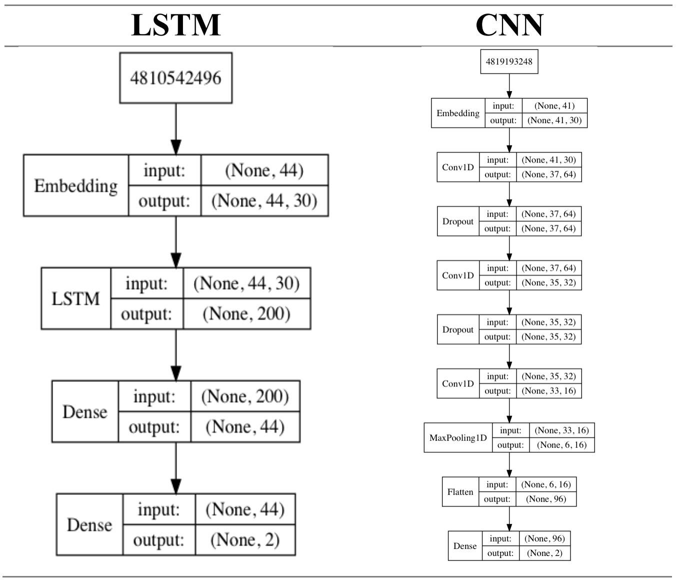
2.3.1 Supervised Learning

The following classifiers are implemented and ready for tuning.

MODEL	***PARAMETERS TO TUNE***
KNN	K-neighbors
Naïve Bayes	Prior-fit
SVM	C – Kernel
Decision Trees	Criterion
Random Forest	N-estimators
Logistic Regression	Penalty – Solver
MLP	Activation Function – Solver
Adaboost	Not Used
Bagging	Not Used

2.3.2 Deep Learning

The model that was adapted is based on RNNs, specifically Long Short-Term Memory Network (LSTM). They used an ensemble of classifiers but we used a striped down version of it by only using one classification phase.
Along to the adapted model, we have implemented a 1 – dimensional convolutional model with max-pooling. It is solely based on intuition and nothing else. They were both built using Keras API with no tuning. To use them we skip the vectorization phase as Keras has its own Embedding layer that does the job for us, both had word vectors of dimension 30. But we must calculate the size of the vocabulary so the count vectorizer came to our aid. Corpus and labels are both one-hot encoded to be fed to the network, data was also padded with the maximum length of the samples.


LSTM
CNN
4810542496
Embedding input: (None, 44) output: (None, 44, 30)
LSTM input: (None, 44, 30) output: (None, 200)
Dense input: (None, 200) output: (None, 44)
Dense input: (None, 44) output: (None, 2)
4819193248
Embedding input: (None, 41) output: (None, 41, 30)
Conv1D input: (None, 41, 30) output: (None, 37, 64)
Dropout input: (None, 37, 64) output: (None, 37, 64)
Conv1D input: (None, 37, 64) output: (None, 35, 32)
Dropout input: (None, 35, 32) output: (None, 35, 32)
Conv1D input: (None, 35, 32) output: (None, 33, 16)
MaxPooling1D input: (None, 33, 16) output: (None, 6, 16)
Flatten input: (None, 6, 16) output: (None, 96)
Dense input: (None, 96) output: (None, 2)

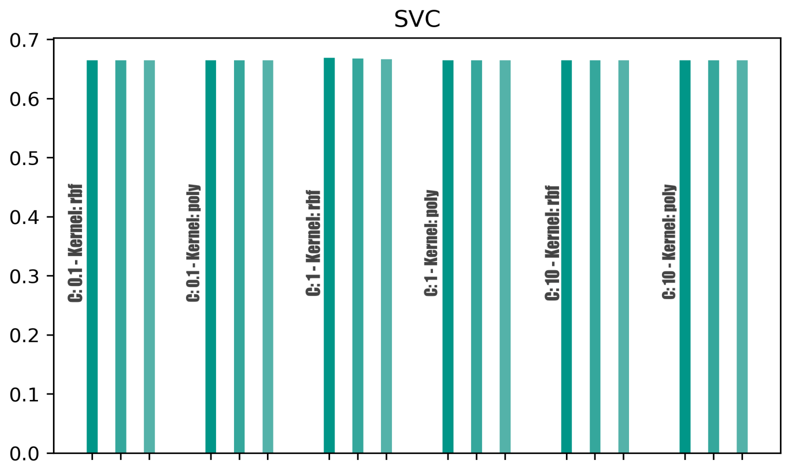
3 Tuning

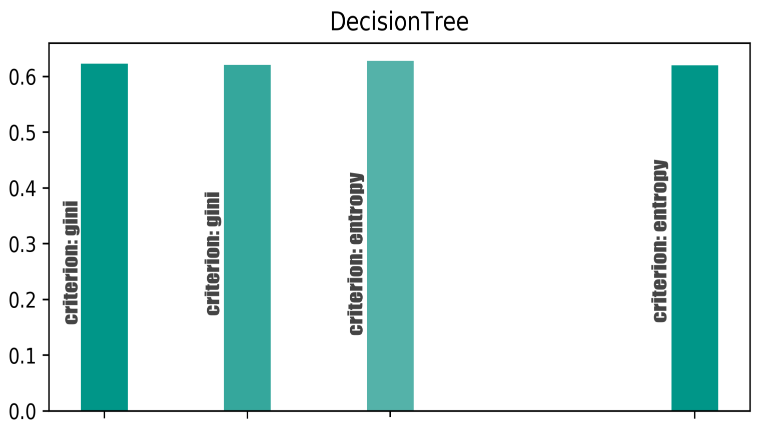
Supervised classifiers were tuned using 3-fold cross validation using GridSearchCV. The following parameters grids were tested. Grid search is the process of testing all possible combination of parameters ranges given to get the best model that resulted in the best mean validation accuracy / least loss.

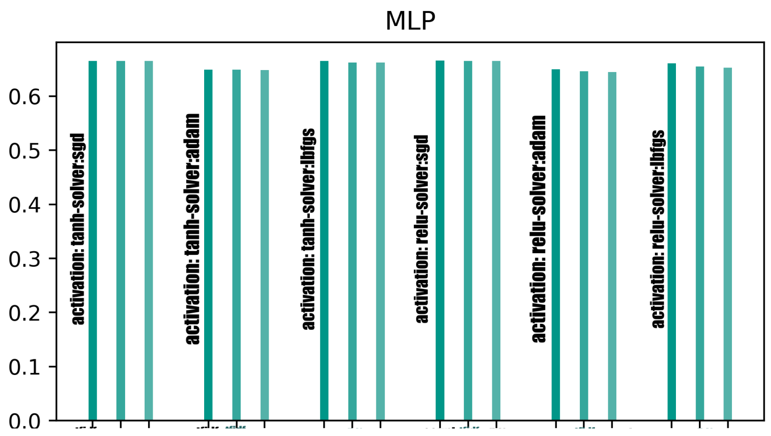
MODEL	***PARAMTERS GRID***
KNN	K: [1, 3, 5, 7]
Naïve Bayes	Prior-Fit: [True, False]
SVM	C [0.1,10,100] – Kernel: [rbf, poly]
Decision Trees	Criterion: [gini, entropy]
Random Forest	N-estimators: [10→200, step:10]
Logistic Regression	Penalty :[l2] – Solver: [sag, lbfgs, newton-cg]
MLP	Activation Function:[tanh, relu] – Solver: [sgf, adam, lbfgs]

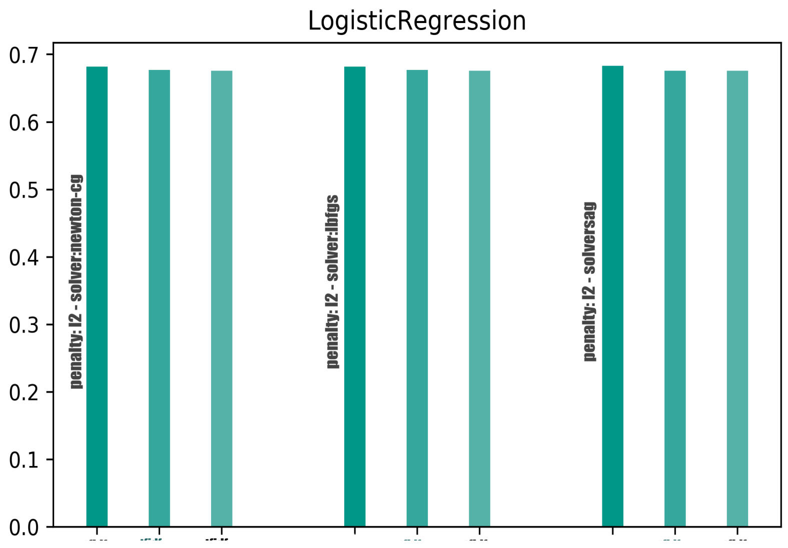
3.1 Tuning Results

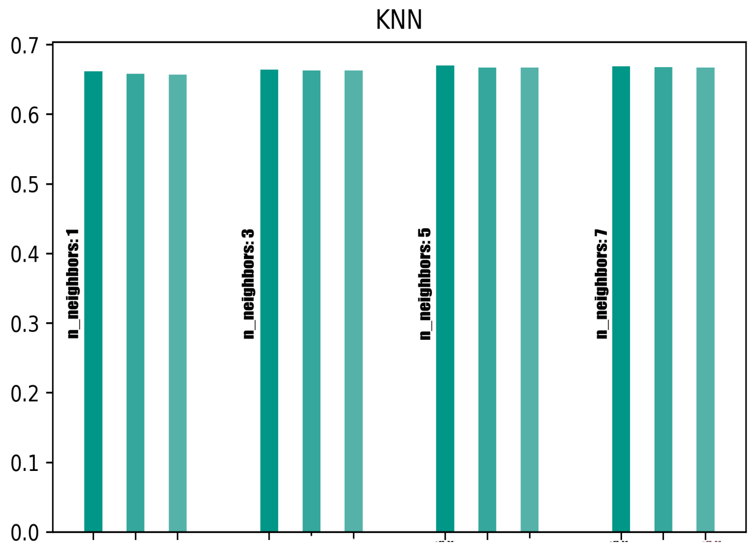
The following graphs represent the best accuracies achieved by each learner's best parameters, keeping in mind that each classifier have been tuned on several combinations of preprocessing & vectorization techniques. Each 3 bars together (2 in Decision Trees) represent the top accuracies [sorted] for a hyper-parameter combination with variations of preprocessing & vectorization techniques.

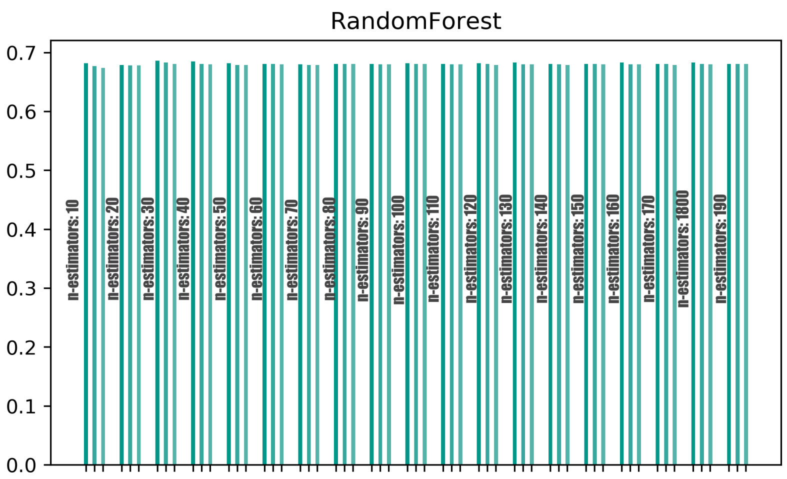


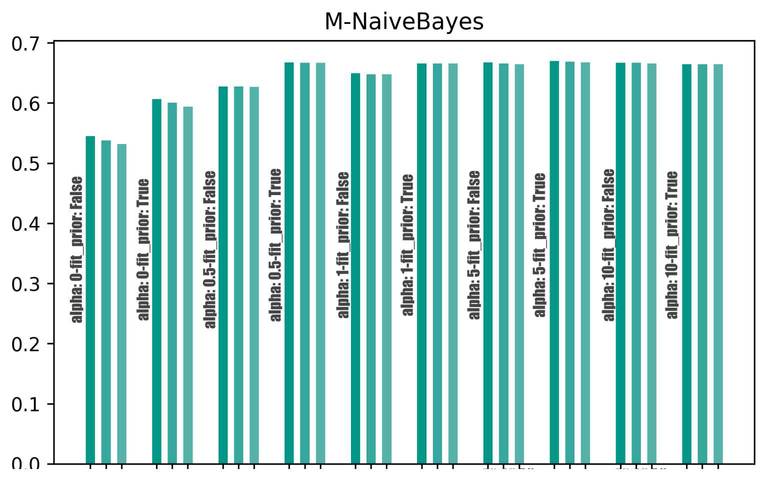












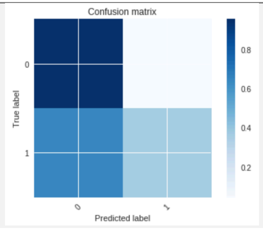
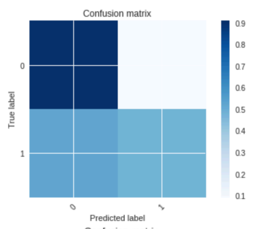
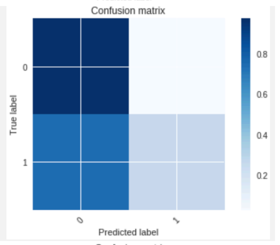
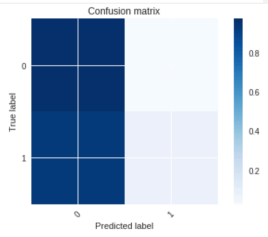
Results for the top 5 leaners with their parameters, preprocessing and vectorization techniques.

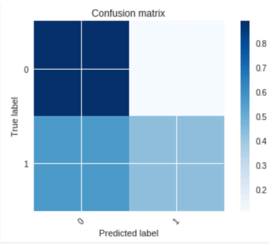
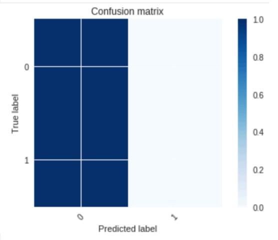
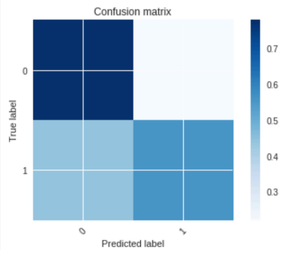
CLASSIFIER	PARAMETERS	PREPROCESSING	VECTORIZATION	VALIDATION ACCURACY
Random Forrest	n-estimators: 30	Lemmatization	Count	0.686
Logistic Regression	penalty: l2 solver: sag	Stopwords removal Lemmatization	Count	0.683
Naïve Bayes	alpha: 5 fit-prior: True	Stopwords removal Stemming	Count	0.67
KNN	k: 5	Lemmatization	TFIDF	0.67
SVC	C: 10 kernel: rbf	Stopwords removal	GloVe	0.669

4 Testing

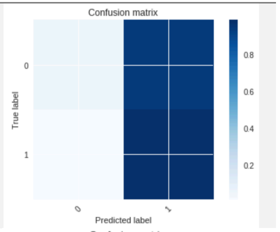
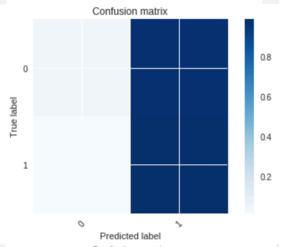
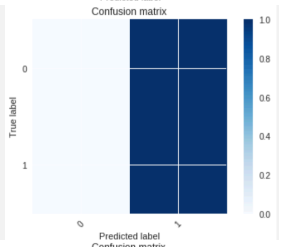
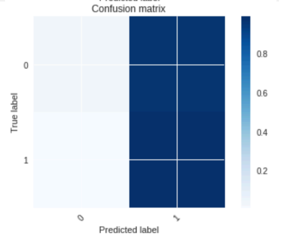
The top 5 models mentioned previously are tested with a 70-30 split on all three subtasks, results were as follows:

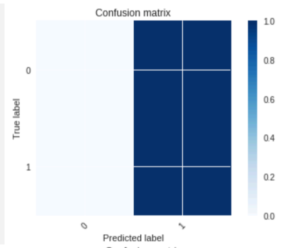
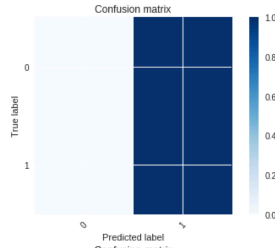
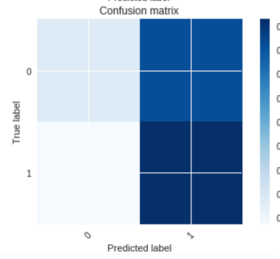
4.1 Subtask A

CLASSIFIER	ACCURACY	CONFUSION MATRIX
Random Forrest	0.759	
Logistic Regression	***0.767***	
Naïve Bayes	0.737	
KNN	0.679	

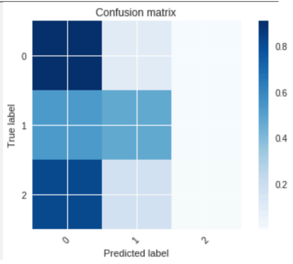
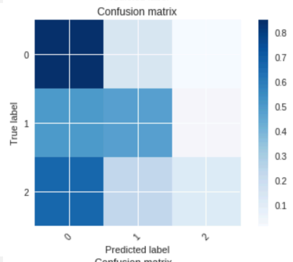
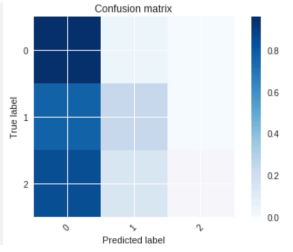
SVC	0.738	
LSTM	0.668	
CNN	0.706	

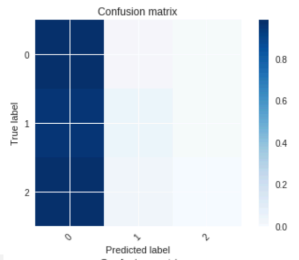
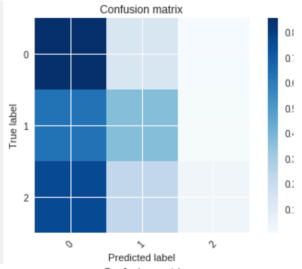
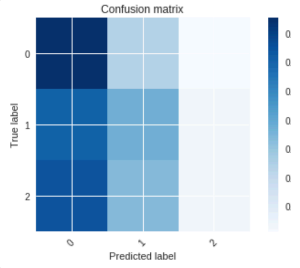
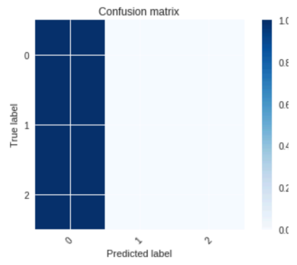
4.2 Subtask B

CLASSIFIER	**ACCURACY**	**CONFUSION MATRIX**
Random Forrest	0.878	
Logistic Regression	0.875	
Naïve Bayes	***0.884***	
KNN	0.878	

SVC	0.883	
LSTM	0.869	
CNN	0.828	

4.3 Subtask C

CLASSIFIER	**ACCURACY**	**CONFUSION MATRIX**
Random Forrest	***0.703***	
Logistic Regression	0.677	
Naïve Bayes	0.671	

KNN	0.631	
SVC	0.641	
LSTM	0.625	
CNN	0.57	

4.4 Results

The best results achieved in all three subtasks are:

Subtask	*Classifier*	*Accuracy*
A	Logistic Regression	0.767
B	Naïve Bayes	0.884
C	Random Forest	0.703

5 Conclusion

Results are kind of random (bad). There's no one model that stands out in all subtask, each has its own best performer. Which is logical because the data isn't that good enough, tweets in general contain too much noise and aren't consistent with each other, a single word could be spelled 10~20 times differently and vectorizers will treat it not as one word which increases the features to work with while not increasing any information gained from such increase. Also, the dataset's variance isn't quite good, more than 65% of the data is labeled NOT, so a model's predictions are mostly NOT as well that explains the accuracy of subtask A, the same goes for subtasks B and C as well. This shows that most classifiers over-fit the training data due to its high variance and unbalanced data.

References

Georgios K. Pitsilis, Heri Ramampiaro and Helge Langseth, '*Detecting Offensive Language in Tweets Using Deep Learning'* URL https://arxiv.org/pdf/1801.04433.pdf

Myan Sherif, Sherine Mamdouh and Wegdan Ghazi, *CTSys at SemEval-2018 Task 3: Irony in Tweets* URL: http://aclweb.org/anthology/S18-1094

Shivam Bansal, *Ultimate Guide to Understand & Implement Natural Language Processing URL https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/*