Vigilante at SemEval-2019 Task 6: How Far Can Models Overcome Data Imbalance

Ioan Budea
Imperial College London
ib2215@ic.ac.uk

Adrian Catană
Imperial College London
ac5915@ic.ac.uk

Abstract

Social media content generation is driven by the constant user growth. With this trend, the amount of hate speech is also increasing, making the necessity of automated methods for offensive language detection more proeminent. This paper presents the empirical results on the OffensEval challenge on identifying and categorizing offensive language in social media. We present the approaches tried, the different architecture of the models and discuss the results obtained and challenges encountered in designing classifiers for this purpose.

1 Introduction

With the emergence of social networking websites, the information posted and shared by users reaches a tremendous audience instantly. However, these websites and technologies have also enabled individuals to disseminate aggressive and harmful content as well, shielded by this layer of indirection. Finding a way to limit the generation and impact of offensive speech has attracted a lot of attention from the research community in recent years (Davidson et al., 2017; Malmasi and Zampieri, 2017). The amount of new data posted every day makes manual monitoring and moderation of content infeasible.

In this paper, we present out findings on the SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media. The competition required building classifiers for three different subtasks: offensive language Identification (Task A - binary classifier to detect offensive and acceptable language), automatic categorization of offense types (Task B - binary classifier to split the offensive content in Task A into targeted and untargeted) and offense target Identification (Task C - ternary classifier for targeted offensive language). The participants were provided with a dataset of 13200 labelled tweets to be used as training data.

In Natural Language Processing (NLP), classical methods required manual feature engineering and used statistical algorithms to perform the majority of the tasks. The new wave of NLP models borrow ideas from deep learning to perform extraction of hierarchical features from the given data. For our submissions, we decided to use Random Forests as an algorithm representative for the first category, and Convolutional Neural Networks and LSTM for the deep learning approaches.

2 Data Processing

The gievn dataset from the competition starting kit contains 13240 tweets: 4400 of them are marked as offensive and 8840 are not offensive. The offensive ones are further split for task B into 524 untargeted and 3876 targeted. Finally the targeted ones are divided for task C into: ...

We have used the holdout method to split these tweets into a training set 8712 and a validation set comprising the remaining examples.

TODO: After we report the results of our validation experiments, we train the classifiers on the entire dataset before submission.

As a small test set we have parsed an additional file present in the starting kit to construct a dataset of 320 tweets.

We convert the tweets to lowercase, remove the following unnecessary space patterns and non-alphanumeric characters from the tweets, and remove the stopwords using the nltk Python package. After applying data augmentation, stemming or lemmatization (described in the subsections below), a vocabulary is built from the words in the training set. Each word in the sentences is then transformed into a number based on the index in this vocabulary. Then, each sentence is padded to

the maximum length of all sentences by filling the vectors with the element 0 which represents the special ¡pad¿ symbol.

2.1 Data augmentation

From the above ratios, we see that the number of examples from each class are not comparable. While the issue is not so proeminent for the training dataset used for task A, this class imbalance might become problematic when investigating targeted offensive language. Therefore, we have decided to apply the following techniques for data augmentation:

- increasing the number of offensive sample by taking an offensive tweet and appending a random harmless tweet to create a new test instance
- increasing the number of untargeted sample by taking an untargeted tweet and duplicating it as another test instance
- Maybe augment for C?

TODO: explain augmentation after split - risk to have data in validation

2.2 Stemming

We have also attempted word stemming for our tasks. Stemming can be defined as a rather crude heuristic process that removes the affixes of words in the hope of bringing the inflectional forms and derivationally related forms of a word to a common base form. In our work we have used the Porter stemming algorithm implemented in the same nltk Python package.

2.3 Lemmatization

While the goal of lemmatization is similar to the case of stemming, the latter method obtains the stem of a word after applying a set of rules without taking into account the part of speech or the context of the word occurrence. Lemmatization obtains the base form of a word known as a lemma of a word which involves reducing the word forms to its root form after doing a full morphological analysis of a sentence. While this technique may work in some application, it can hurt the performance of others. We have attempted this alternative preprocessing because stemming usually manages to increase recall while harming the precision of the classifiers, so a more informed decision to transform words may provide a better result. For this work, we have used a lemmatizer based on the WordNet lexical database of English words.

2.4 Slang

3 Classifiers

Other work on hate speech in tweets (Badjatiya et al. (2017)) has been conducted using deep learning techniques. Our experiments use two of the methods proposed: Convolutional Neural Networks and Long short-term memory, comparing them against the general framework of Random Forests. WHAT? Moreover, they concluded that embeddings learned from deep neural network models when combined with gradient boosted decision trees led to best accuracy values, which significantly outperforms the existing methods.

In this section we describe the classifiers tested, the framework used for the implementation and the hyperparameters that can be tuned for each approach.

3.1 Random Forest

Random Decision Forests [] are an ensemble learning technique that constructs multiple decision trees using the training data and looks at the mode of the classes to decide the prediction when classifying new data.

The parameters that can be modified are the number of estimators (i.e. the number of trees in the forest), the maximum depth of each tree (having a limit is a good technique for preventing overfitting) and the criterion for each spilt (a choice of either the entropy or gini measure).

Using the RandomForestClassifier from the sklearn Python package, we have built random forests of 30 trees with a maximum depth of 80 nodes for each of the three tasks.

How dafuq we have found the values?

3.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) [] is a feed-forward neural network consisting of the typical input and output layers, but with the hidden layers having additional types: convolution and pooling. The idea of the convolutional layers is to enable the network to automatically identify and extract features from the input space, while pooling layers can be introduced to reduce the dimentionality of the problem space. CNNs are mostly used in applications such as image and

sound recognition, but they can be suited to particular NLP tasks as well. The intuition is that convolutional filters can correspond to several different semantic patterns [Jacovi, 2018] and thus, in our context of offensive language classification, CNNs can extract particular word combinations.

Our CNN is implemented using the PyTorch framework[]. Once the input vectors are built using the index of words in the known vocabulary, these will be used as input to our CNN. The first layer is an embedding layer that transforms each word into a continuous vector space of given dimension (Mikolov). Thus, the word embeddings will be contextualised and learned from the given examples. Next, the output of this embedding layer is fed into a convolutional layer, followed by a ReLU activation function, a max pooling layer and a dropout layer. At the end, a linear layer is applied to bring the result to either a single output value which will be transformed into a probability value for the binary classification task or a vector representing the corresponding outputs of all the classes (i.e. three classes for task C).

The hyperparameters that can be modified are: the number of layers, the dimension of the convolution kernels, the dropout probability, the optimization algorithm and learning rate used.

We have opted for a shallow network architecture because the tweets have a small maximum length, so the number of dimensions doesn't need to be reduced dramatically. By varying the window size from the text taken into account in our convolutional layer we found that ... yields the best results. The CNN was trained using the Adam optimizer [] to accelerate convergence during training with a learning rate of 0.0008 found using grid search.

3.3 Long-Short Term Memory

Recurrent Neural Networks (RNN) have been shown to produce very strong results for language modeling []. They are suited for language tasks because they take into account the whole input sequence, storing the previous results in a hidden state within the nework architecture. Long-short term memory [] is a very popular type of RNNs, proposed to address the problem of vanishing and exploding gradients and make RNN training feasible.

4 Results

Below we present our best runs for the algorithms at each individual task.

4.1 Validation Experiments

100 epochs

RF results will predict only one label CNN with data augmentation, split, stemming: Epoch: 99 — Train Loss: 0.528 — Train Acc: 73.72% — Val. Loss: 0.555 — Val. Acc: 71.25% — — Test Loss: 0.583 — Test Acc: 67.19% 58 Test: Recall: 0.75, Precision: 0.40, F1-measure: 0.52

80 epochs

Stemming, split, augment Test Acc: 71.79Test: Recall: 0.14, Precision: 0.40, F-measure: 0.21

Lemmatization, split, data augmentation Test Acc: 71.79Test: Recall: 0.07, Precision: 0.25, F-measure: 0.11

4.2 Submission Results

4.3 Discussion and Interpretation

Better if we would have used the whole English vocabulary or a bigger subset of it for our word indices since we cannot be ceratin that words in the training set are representative for the test set as well, and this alternative approach would provide a better way of treating new, unseen words

5 Conclusion

Task A	baseline	stemming	lemmatization
Random Forest	Test Acc: 72.81%	Test Acc: 70.31%	Test Acc: 75.31%
CNN	Test Acc: 82.50% F1-measure: 0.49	Test Acc: 85.94% F1-measure: 0.65	Test Acc: 81.25% F1-measure: 0.41
LSTM	bold	bold	bold

Table 1: Task A experimentation results.

Task B	baseline	stemming	lemmatization
Random Forest	Test Acc: 53.24%	Test Acc: 51.94%	Test Acc: 53.24%
CNN	Test Acc: 68.83% F1-measure: 0.74	Test Acc: 58.44% F1-measure: 0.67	Test Acc: 63.64% F-measure: 0.70
LSTM	bold	bold	bold

Table 2: Task B experimentation results.

Task C	baseline	stemming	lemmatization
CNN	Test Acc: 71.79% F1-measure: 0.12	Test Acc: 69.23% F1-measure: 0.11	Test Acc: 69.23% F1-measure: 0.11
LSTM	bold	bold	bold

Table 3: Task C experimentation results.