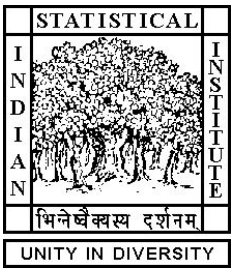
# Classification of Micro-Blog Texts




Bihan Sen

Roll Number: CS1705

Computer Vision and Pattern Recognition Unit

Indian Statistical Institute, Kolkata

*Supervisor*

Dr. Mandar Mitra

In partial fulfillment of the requirements for the degree of

*Master of Technology in Computer Science*

June 28, 2019

# CERTIFICATE

This is to certify that the dissertation titled **"Classification of Micro-blog Texts"** submitted by **Bihan Sen**, Roll Number **CS1705** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** embodies the work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

---

**Mandar Mitra**
Associate Professor,
Computer Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata-700108, India.

# Acknowledgements

First of all, I would like to express my sincere gratitude towards my supervisor Dr. Mandar Mitra for encouraging me to do this work, supporting me throughout this thesis and helping me with the challenges I had faced. He has always helped me gain the courage to go along with this work.

I am grateful to Dr. Dwaipayan Roy for providing necessary inputs and being part of healthy discussions whenever in doubt.

Lastly, I would also like to thank my friends and most importantly my parents for supporting me throughout the duration and having faith in me.

# Abstract

Classification of micro-blog texts is a very common task for sentiment analysis, user opinion mining, product review analysis, crisis managements, identifying offensive and hate speech propagation across social media, restricting unnecessary expansion of fake news and rumors etc. In this dissertation, we consider two problems from this domain: (i) classification of tweets during crisis scenarios like natural disasters, terrorist attacks etc and (ii) identifying offensive tweets. We tried both statistical and deep learning approaches. Datasets from the TREC-IS 2018 and 2019 tasks, and OLID from OffenseEval workshop were used for our experiments. The first task is formulated as a multi-label classification task, while the second is a binary classification problem. Our results suggest that preprocessing of social media text is very crucial for classification. We also conclude that Deep Learning approaches do not always outperform traditional learning. We also took part as an active participant in the TREC-IS 2019A task. Out of all 34 submissions from across the world, one of our submissions achieved the highest macro-averaged F-1 score on this task (0.1969) and outperformed the second highest score (0.1556) by a substantial margin.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

Social networking sites like Twitter and Facebook are now a leading source of news about current events. events throughout the world are featured in these social media sites almost immediately. Because these websites are used and followed by general users around the globe, they have a higher reach than online newspapers, regular blogs, magazines and traditional media. Every second, on average, around 6,000 tweets are tweeted on Twitter, which corresponds to over 350,000 tweets sent per minute, 500 million tweets per day and around 200 billion tweets per year. It is estimated in 2019, 87,500 individuals are on Twitter every minute. 71% of these users use the platform for news updates. Similarly, during December 2018, 1.52 billion people on average log onto Facebook daily and are considered daily active users (Facebook DAU)[1]. Traditional text processing tasks, such as retrieval and classification, are particularly challenging when studied in the context of micro-blogs because of the extreme brevity of individual "documents". During the course of this dissertation, we will consider the problem of text classification applied to micro-blogs.

[1] `https://www.statista.com/statistics/346167/facebook-global-dau/`

## 1.2 Objectives and Contributions

From recent events, it has become evident that social media texts (mainly tweets) have played a vital role during crisis scenarios such as earthquakes, floods, explosions, terrorist attacks etc. With this rise of social media, emergency service operators are now expected to monitor such channels and answer questions from the public. However, crisis management personnel do not have adequate tools or manpower to effectively monitor social media, due to the large volume of information posted on these platforms. During the Nepal earthquake, for example, 33,610 tweets were reportedly posted between April 25th and May 28th by people on the ground in Nepal [1]. These tweets contained a great deal of useful information which could be vital for rescue operations, but 33,610 tweets is simply too many for operators to process manually. There is thus an urgent need for tools that automatically categorize, cross-reference and verify the information available via micro-blogs during disasters.

On a related note, user opinions about recent events such as political decisions, supporting similar views etc often contain offensive words and hate speech. Davidson et. al. [2] defined hate speech as *language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group.* However, this definition does not include all instances of offensive languages. People often use terms that are highly offensive to certain groups but in a qualitatively different manner. For example some African Americans often use the term *n*gga* in everyday language online, people use terms like *h*e* and *b*tch* when quoting rap lyrics, and teenagers use homophobic slurs like *f*g* as they play video games [2]. Such language is prevalent on social media, making this boundary condition crucial for any usable hate speech detection system. The identification and filtration of such offensive posts and/or tweets is very time-consuming and as it can cause post-traumatic stress disorder-like symptoms to human annotators, there have been many research efforts aiming at automating the process.

In this dissertation, two important tasks have been studied: *Building Effective Disaster Management Systems from Tweets*, and *Offensive content Identification from Tweets.* We build and test our systems using standard, benchmark datasets

that are publicly available.

## 1.3 Overview of the Thesis

In Chapter 2, we discuss the Text Categorization problem and some features and characteristics of micro blog texts. Section 2.2 outlines the most popular text categorization algorithms. We discuss multi-label algorithms as well as recent deep learning techniques. In Chapter 4, we describe the problem statement along with the datasets used in our experiments. In Chapter 5 and 6, we present our approaches to the tasks at hand. Experimental results are discussed with the approaches. We enlist the outputs of different approaches and conclude how different methods perform. Lastly, in Chapter 7, we conclude our overall work.

# Chapter 2

# Short Text Classification

In this chapter, we briefly review text categorization *a.k.a* classification, as well as different machine learning and state-of-the-art approaches for text classification. We also discuss the problems of handling short texts in these tasks.

## 2.1 Text Classification

Content-based document management tasks (collectively known as *information retrieval* - IR) have gained a prominent status in the information system due to huge availability of documents in digital form. *Text Classification* a.k.a *Text Categorization* is one such task in which labelling of a document is performed by analyzing the linguisitic contents on the document. Due to rise of *Machine Learning* - ML in the '90s, in which an automatic classifier is trained from a set of pre-classified documents, text classification gained more popularity as there is no intervention needed from either the Knowledge Engineers or the Domain Experts.

### Definition of Text Classification

Sebastiani [3] defined Text Categorization as the task of assigning a Boolean value to each pair $(d_j, c_i) \in D \times C$, where $D$ is the domain of documents and $C = \{c_1, \ldots, c_{|C|}\}$ is a set of predefined categories. A value of $T$ assigned to $(d_j, c_i)$ indicates a decision to file $d_j$ under $c_i$, while a value of $F$ indicates a

decision not to file $d_j$ under $c_i$. More formally, the task is to approximate the unknown *target function* $\hat{\phi} : D \times C \rightarrow \{T, F\}$ by the means of a *decision function* $\phi : D \times C \rightarrow \{T, F\}$ called a *classifier* (a.k.a *hypothesis*, or *model*) such that $\hat{\phi}$ and $\phi$ "coincide as much as possible."

## 2.2 Text Classification : A Brief Study

Yang [4], Sebastiani [3], Delgado et. al. [5] and other authors have summarized, analyzed and compared different text classification techniques, starting from rule based to statistical approaches and some recent Deep Learning techniques for text classification. In this section we briefly discuss a subset of classifiers and different types of classifications that are used in text classification.
Depending on the application task in hand, different constraints on Classification can be imposed, producing mainly two kinds of Classification. For instance we might need that, for a given integer $k$, exactly $k$ (or $\leq k$, or $\geq k$) elements of $C$ be assigned to each document $d_j \in D$. The case in which exactly one category must be assigned to each $d_j \in D$ is often termed as **Single Label Classification**, whereas the case in which, a document $d_j \in D$ can be assigned to any random number of labels from $C$ is called **Multilabel Classification**. A special case of Single Label Classification is **Binary Classification** in which the task is to categorize each $d_j \in D$ to either category $c_i$ or its complement $\bar{c_i}$.

### 2.2.1 Classification Algorithms

Machine learning approaches are used in Text Classification problems. Recent researches in Deep Learning algorithms have brought a breakthrough in image processing, computer vision, text domains etc.

Logistic Regression (LR) is one of the earliest methods for classification tasks. It was developed by statistician David Cox in 1958. The aim of Logistic Regression is to predict the conditional probability $P(y|x;\theta)$, where $\theta$ is the model parameter. The hypothesis of logistic regression tends it to limit the output

between 0 and 1, with the S-shaped *sigmoid function*,

$$h_\theta(x) = \frac{1}{1 + e^{\theta^T x}}$$

The cost function for LR is given by,

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

If $y = 1$, the cost approaches to 0 when $h_\theta(x)$ approaches to 1. Conversely, the cost increases up to infinity when $h_\theta(x)$ approaches to 0. Similar intuition applies when $y = 0$. Hence the cost function is defined in a way such that a big penalty is given when the prediction is far from actual output. As the cost function is convex in nature, the parameter $\theta$ for which it attains the minimum value can be determined using the gradient descent algorithm.

K Nearest Neighbour (KNN) [6] is a non-parametric, Instance Based Learning technique used for classification. Given a test document $x$, the algorithm finds $k$ most similar documents among the training set. This similar documents are determined by obtaining $k$ nearest neighbours of $x$ in the feature space. The test document is classified by the plurality of votes of its neighbours. An useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $\frac{1}{d}$, where $d$ is the distance to the neighbour.

Support Vector Machine (SVM) is a non-probabilistic large margin classifier. It constructs a hyperplane in a high- or infinite-dimensional space, which separates the positive from the negative training example. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class, since in general the larger the margin, the lower the generalization error of the classifier. Using parameter $w$ and $b$, the

linear classifier can be expressed as,

$$h_{w,b}(x) = g(w^T x + b)$$

where $g(z) = 1$ if $z \geq 0$ and $-1$ otherwise. Given the choice of $g$ we can normalize $w$ such that for the nearest data point $x_n$, $|w^T x_n + b| = 1$. Now the distance between $x_n$ and the hyperplane is given by,

$$|\frac{w}{\|w\|}(x - x_n)| = \frac{1}{\|w\|}$$

. After some mathematical equivalences, the optimization problem becomes,

$$\text{minimize } \frac{1}{2} w^T w \text{ subject to } y_i(w^T x_i + b) \geq 1 \text{ for } i = 1 \dots N$$

This is easily solved by forming the dual convex problem using Lagrange's Method. Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. The original finite-dimensional space can be mapped into a much higher-dimensional space, presumably making the separation easier in that space, by using a method called 'Kernel Trick'.

### Deep Learning Approaches

Recurrent Neural Network (RNN) is basically a 3-layer neural network, which repeatedly uses its layers at each time step for accommodating a sequence. If $x_t$ is taken as the input to the network at time step $t$ and $s_t$ represents the hidden state at the same time step. Calculation of $s_t$ is based as per the equation:

$$s_t = f(Ux_t + Ws_t)$$

Thus, $s_t$ is calculated based on the current input and the previous time steps hidden state. The function $f$ is taken to be a non-linear transformation such as $tanh, ReLU$ and $U, V, W$ account for weights that are shared across time. Due to variable and long length of sequences, in practice, simple RNN networks suffer from the infamous *vanishing gradient* problem, which makes it really hard to

learn and tune the parameters of the earlier layers in the network.

Long Short-Term Memory (LSTM) [7] has additional "forget" gates over the simple RNN. Its unique mechanism enables it to overcome both the *vanishing* and *exploding gradient problem*. Unlike the vanilla RNN, LSTM allows the error to back-propagate through unlimited number of time steps. Consisting of three gates: input, forget and output gates, it calculates the hidden state by taking a combination of these three gates as per the equations below:

$$f_t = \sigma(W_f[x_t; h_{t-1}] + b_f) \tag{2.1}$$

$$i_t = \sigma(W_i[x_t; h_{t-1}] + b_i) \tag{2.2}$$

$$o_t = \sigma(W_o[x_t; h_{t-1}] + b_o) \tag{2.3}$$

$$\hat{C}_t = \sigma(W_c[x_t; h_{t-1}] + b_c) \tag{2.4}$$

$$c_t = f_t * c_{t-1} + i_t * \hat{C}_t \tag{2.5}$$

$$h_t = o_t * tanh(c_t) \tag{2.6}$$

where $[\mathbf{u}; \mathbf{v}]$ signifies concatenation of vector $\mathbf{u}$ and $\mathbf{v}$. A variation of LSTM is Bidirectional LSTM (Bi-LSTM) which processes the text from both left-to-right and right-to-left direction and the two outputs are concatenated together to form a single output.

Attention is a mechanism [8] mainly used in neural encoder-decoder systems. Attention-based mechanism is motivated from that, instead of decoding based on the encoding of a whole and a fixed-length sentence during one pass of neural network-based machine translation, one can **attend** a specific part of the sentence. Hence the final context vector is basically some weighted sum of the hidden states (annotations) $(h_1, h_2, \ldots h_T)$:

$$c_i = \sum_{i=1}^{T} \alpha_{ij} h_j$$

The weight $\alpha_{ij}$, also called *attention score*, of each annotation $h_j$ is computed by,

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

where,

$$e_{ij} = a(s_{i-1}, h_j)$$

is an alignment model which scores how well the inputs around position $j$ of target and the output at position $i$ of the source match. In case of text-classification, this alignment model is applied on the encoder RNN itself, computing how importance each hidden state should be given to get the final context vector.

Convolutional Neural Network (CNN), gaining a huge success in the image and computer vision domain as an automatic feature extraction method, attracted the NLP researchers to extract features from text. A work by Kim [9] has been well studied and produced excellent results using CNN for sentence classification tasks on benchmark datasets. A following work by Zhang and Wallace [10] has explained the feature extraction mechanism of CNN for text classification tasks and effect of different parameters.

A tokenized sentence is converted to a sentence matrix, the rows of which are word vector representations of each token. If the dimension of the word vectors is denoted by $d$ and length of a given sentence is $s$, then the dimension of the sentence matrix is $s \times d$. In text applications there is inherent sequential structure to the data. Because rows represent discrete symbols (namely, words), it is reasonable to use filters with widths equal to the dimension of the word vector. If there is a filter parameterized by the weight matrix $W$ with region size $h$ (height of the filter); $W$ will contain $h.d$ parameters to be estimated. This sentence matrix is denoted by $A \in \mathbb{R}^{s \times d}$ , and $A[i : j]$ represents the sub-matrix of $A$ from row $i$ to row $j$. The output sequence $o \in \mathbb{R}^{s-h+1}$ of the convolution operator is obtained by repeatedly applying the filter on sub-matrices of $A$:

$$o_i = W.A[i : i + h - 1]$$

where $i = 1 \dots s - h + 1$, and . is the dot product between the sub-matrix and the filter (a sum over element-wise multiplications). A bias term $b \in \mathbb{R}$ is added and an activation function $f$ to each $o_i$, inducing the feature map $c \in \mathbb{R}^{s-h+1}$ for this filter

$$c_i = f(o_i + b)$$

Multiple filters may be used for the same or different region size to learn features. The dimension of the feature map generated by each filter will vary as a function of the sentence length and the filter region size. A pooling function is thus applied to each feature map to induce a fixed-length vector. A common strategy is 1-max pooling, which extracts a scalar from each feature map. Together, the outputs generated from each filter map can be concatenated into a fixed-length, 'top-level' feature vector.

In text classification task, the final output of the LSTM or a CNN, also referred as *feature vector* or *context vector* (in case of seq2seq model) of the text that has been processed, is passed through a *fully-connected layer* and the number of nodes in the output layer is kept as the number of the class (1 in case of binary) specified in the task, followed by a sigmoid (binary classification) or a softmax activation, to transform the score values into a probability distribution across classes. Under this construction the loss is calculated as, Cross Entropy Loss

$$CE = -\sum_{i=1}^{C} y_i \log(s_i)$$

where $y_i$ and $s_i$ are the ground truth and the probability score respectively.

Due to huge success of transfer learning in NLP, in 2018, deep architectures for multi-tasking are now an interesting point of research. Google's BERT [11], AllenNLP's ElMo [12], OpenAI GPT2 [13], fastai's ULMfit [14] etc these new multi-tasking architectures are pre-trained and can be plugged to various NLP tasks such as sentence classification, sentence pair classification, question answering etc. It has been claimed that BERT outperform all the others, as of now. BERT is based on another architecture named Transformer [15]. Briefly, A Transformer, primarily developed for machine translation, takes a fixed size input, passed through a stacked layer of Encoders. Each Encoder consists of a self-attention layer and a feed forward neural network. The self-attention layer calculates how each token of input sentence is dependent on all other tokens of the same. The final encoded input is passed through another stacked layer of Decoders. Each decoder is again similar to encoder except that it has one

encoder-decoder attention layer along with self attention. This incorporates the similar idea from seq2seq model implying how important a source sentence token is to target sentence token.

BERT-base consists of 12 layers, and BERT-large has 24 layers of Transformer Encoders, with a target of building a language model which contains both directional contexts, called Masked Language Model. Beyond masking 15% of the input, BERT also mixes things a bit in order to improve how the model later fine-tunes. Sometimes it randomly replaces a word with another word and asks the model to predict the correct word in that position. Since BERT is a general language model, the architecture can be tweaked for different NLP tasks such as sentence classification, where it reportedly outperformed most of the other methods on some benchmark datasets.

#### 2.2.1.1 Multilabel Classification

Multilabel Classification has been a very renowned topic in the last decade. Madjarov et. al. [16] have contributed an extensive study of the available multi-label classification algorithms and their effectiveness among 11 popular benchmark multi-label datasets. Tsoumakas and Katakis [17] have presented the first overview of multi-label learning, where they divided the methods into two main categories: *Problem Transformation* and *Algorithm Adaptation*. Madjarov et. al. [16] have added another category named *ensemble methods* to the existed categories for completeness. Here we discuss a very brief overview of such algorithms.

**Problem Transformation Methods**

Problem Transformation methods transform a multi-label learning to one or more single label classification problem. The algorithms discussed in section 2.2 can be used as a base classifier here.

*Binary Relevance* (BR) [17] is the simplest strategy for problem transformation, this method uses one-vs-rest technique for each of the labels individually, transforming into $L$ binary classification problems, where $L$ is the number of labels. For each of the labels, a classifier is built using all relevant examples as positive samples and rest negative. During prediction, each classifier predicts

whether a test sample is relevant for the corresponding label or not. The main drawback of BR is that it does not take into account any label dependency and may fail to predict some label combinations if such dependence is present. However, BR presents several obvious advantages: (1) any binary learning method can be taken as base learner; (2) it has linear complexity with respect to the number of labels; and (3) it can be easily parallelized.

*Classifier Chain* (CC) [18] method involves $L$ binary classifiers, uses binary relevance predictions as extra input attributes for the following classifier, cascaded along a chain. Hence the hypothesis can be formulated as,

$$h_{CC}(\mathbf{x}) = [h_1(x), h_2(x, h_1(x)), \ldots, h_L(x, h_1(x), \ldots, h_{L-1}(x, \ldots))]$$

The prediction of each base classifier $h_j(\mathbf{x}, \ldots)$ only needs to be evaluated per test instance. Each individual classifier may be expressed as,

$$\hat{y_j} = h_j(\mathbf{x}, \hat{y}_i, \ldots, \hat{y}_{j-1}) = \underset{y_j \in \{0,1\}}{\operatorname{argmax}}\, p(y_j|\mathbf{x}, \hat{y}_1, \ldots, \hat{y}_{j-1})$$

and we obtain predictions in order $\hat{y}_1, \ldots, \hat{y}_L$ Along with tractable and approximate search methods for inference, a main focus in the development of CC methods is the order of the chain, to exploit the 'label dependence'.
A variation of CC, namely *Probabilistic Classifier Chain* (PCC) [19] the conditional probability $P(\mathbf{y}|\mathbf{x})$ can be computed by the chain rule of probability,

$$P(\mathbf{y}|\mathbf{x}) = p(y_1|\mathbf{x}) \prod_{j=2}^{L} p(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})$$

One can simply notice that CC is the deterministic approximation of PCC, in the sense of using $\{0, 1\}$ -valued probability.

*Label Power-set* (LP) [20] is another problem transformation method. The basis of this method is to combine entire label set into atomic labels to form a single label problem. To achieve this, the candidate single labels can be the all possible subsets of the label set, thus taking the label correlation into account.
Since the space of possible label subset can be very high, a variation of this

was proposed called, *Pruned Set* (PS). In this variation, a pruning parameter $p$ is specified, subsets whose occurrence is lesser than and equal to $p$ are pruned from the all possible label set. Formally, $(d_i, S_i) \in D$, the entire training set, where count for $S_i$ i.e., $c > p$ are added directly to new pruned dataset $D'$, where $S_i \in 2^Y$ and $Y$ is the label set. Information is saved by taking each possible subset of those pruned label set, having count greater than $p$.

*RAndom k-labELsets* (RAkEL) [21] is an ensemble method, that draws random $m$ subset of label set with cardinality $k$, and trains a label power-set (LP) classifier using each set of labels. A simple voting process determines the final set of labels for a test example. RAkELd is a variation of this algorithm, that draws random $m$ label subsets that are disjoint.

**Algorithm Adaptation Methods**

Multilabel kNN (ML-kNN) [22] is an extension of single label kNN algorithm. It assigns a label to a test example by using a maximum a *posteriori* (MAP) method from the statistical information gained from training data. More formally, for each training example $x$, let $N(x)$ be the $k$ Nearest Neighbours of $x$, a *membership counting vector* can be defined as

$$\vec{C}_x(l) = \sum_{a \in N(x)} \vec{y}_a(l) \; l \in Y$$

where $\vec{y}_a(l) = 1$ if $a$ has the label $l$ and $\vec{C}_x(l)$ counts the number of neighbours of $x$ belonging to the $l$-th class.
For each test instance $t$, ML-kNN first identifies its $k$ nearest neighbours, $N(t)$. Let $H_1^l$ be the event that $t$ has label $l$, and $H_0^l$ denotes the opposite. And also, let $E_j^l$ ($j \in \{0, 1, \ldots k\}$) denote the event that, among $k$ nearest neighbours of $t$, there are exactly $j$ instances having label $l$. Therefore, based on the membership

counting vector $\vec{C}_t$, the category vector is determined as,

$$\vec{y}_t(l) = \underset{b\in\{0,1\}}{\operatorname{argmax}} P(H_b^l | E^l_{\vec{C}_t(l)}) \; l \in Y \tag{2.7}$$

$$= \underset{b\in\{0,1\}}{\operatorname{argmax}} \frac{P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l)}{P(E^l_{\vec{C}_t(l)})} \tag{2.8}$$

$$= \underset{b\in\{0,1\}}{\operatorname{argmax}} P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l) \tag{2.9}$$

*Predictive Clustering Tree* (PCT) [23] are decision trees viewed as a hierarchy of clusters. The root node corresponds to one cluster containing all the data, which is recursively partitioned into smaller clusters. PCTs are constructed using a standard top down induction of decision trees (TDIDT) [24] algorithm. The heuristic that is used for selecting the test is reduction in variance caused by partitioning the instances. Maximizing the variance reduction maximizes cluster homogeneity and improves predictive performance. The main difference between the algorithm for learning PCTs and a standard decision tree learner is that the former treats the variance function and the prototype function that computes a label for each leaf as parameters that can be instantiated for a given learning task. For the multilabel classification case, the variance function is computed as the sum of the entropies of class variables, i.e., Var(E) = $\sum_{i=1}^{t} Entropy(E, y_i)$. The prototype function returns a vector containing the majority class for each target attribute.

Random Forest over Predictive Clustering Tree (RF-PCT) is an ensemble method that uses PCT as base classifier. The diversity among base classifier is obtained using bagging and additionally changing the feature set during the learning. At each node in the trees, a random subset of input attributes is taken, and the best feature is selected from the subset. The number of attributes that are retained is given by a function $f$, for building random forests, $f(x)$ is set to $\lfloor log_2(x) + 1 \rfloor$.

# Chapter 3

# Related Works

## 3.1 Past Works in Building Effective Disaster Management Systems from Tweets

As twitter became a useful tool regarding disaster management after the Nepal Earthquake in 2015, many researchers have tried to formalize the problem and approached with various ways. The datasets were built by different researchers using various crowd sourcing platforms such as MicroMappers [1], CrowdFlower [2] etc. From a huge collection of related studies we summarize the important works here. Muhammad Imran et. al. [25] gathered tweets for different crisis scenarios such as earthquake, typhoon, volcano eruption, landslide, floods, war & conflict etc and they categorized the 52 Million tweets into 19 different classes. They have made their data and contributions publicly available [3] for research purposes. Authors have prepared baseline using SVM, Naive Bayes and Random Forest classifiers on each events separately. In the IRMiDis Track of FIRE, 2018 [4], the organizers have annotated and open sourced 50,000 Tweet messages and 6,000 news articles where the task was to identify *fact-checkable* tweets and retrieve relevant news articles to support them. In SMERP 2018,[5] The primary

[1] http://www.micromappers.org
[2] http://www.crowdflower.com
[3] https://crisisnlp.qcri.org/lrec2016/lrec2016.html
[4] https://sites.google.com/site/irmidisfire2018/
[5] https://www.cse.iitk.ac.in/users/kripa/smerp2018/

focus of the workshop was multi-modal and multi-view information retrieval i.e., developing methods for aggregating information from multiple online and offline data sources (including text, images, and video) for Emergency Relief and Preparedness.

## 3.2 Past Works in Offensive and/or Hate Speech Identification in Tweets

From aggression, cyber bullying to abusive, toxic comment identification, there have been a lots of research on hate speech and offensive content detection in text. However identifying offensive content and hate speech is not completely similar. Kwok and Wang [26] on their work *Locate the Hate : Detecting Tweets against Black* identified presence of offensive word may lead to mis-classify a tweet as a hate speech with bag of word approach. According to their experiment 86% of times, a tweet is classified as hate speech, due to presence of offensive words. The difference between hate speech and other offensive language is often based upon subtle linguistic distinctions, for example tweets containing the word *n*gger* are more likely to be labeled as hate speech than *n*gga*. Many can be ambiguous, for example the word *gay* can be used both pejoratively and in other contexts unrelated to hate speech. Davidson et. al. [2] presented the **hate speech detection** dataset with over 24,000 English tweets labeled as non offensive, hate speech, and profanity. Their work also built a strong baseline with logistic regression, with some feature engineering. In TRAC 2018, the task of **Aggression Identification** [27], participants were provided 15,000 Facebook posts and comments in English and Hindi. For testing two different sets, one from Facebook and one from Twitter were used. The main target of the task was to classify those posts into one of the three categories: non-aggressive, covertly aggressive, and overtly aggressive. The GermEval4 [28] shared task focused on offensive language identification in German tweets. A dataset of over 8,500 annotated tweets was provided for a course-grained binary classification task in which systems were trained to discriminate between offensive and non-offensive tweets. An open competition at

Kaggle, namely, **The Toxic Comment Classification Challenge** [1], provided participants with comments from Wikipedia organized in six classes: toxic, severe toxic, obscene, threat, insult, identity hate.

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

# Chapter 4

# Problem Statement, Dataset Information and Evaluation Measures

## 4.1 Problem Statement

As a part of this dissertation, we took part in the most recent track of TREC, namely Incident Stream (TREC-IS v1.5) [1], in which the organizers have released a collection of around 18,991 tweets related to different events such as, 'TyphoonHagupit', 'NepalEarthquake' and assigned them to 24 different categories. Each tweet may belong to more than one such categories. Hence the task is a multi-label multi-class classification task. Along with this, given a priority value for a tweet in the training data (Low, Medium, High or Critical) another sub-task is to predict 'criticality score' for the tweet (between 0-1), that signifies how actionable a tweet is. We formulate this sub-task as a classification problem.
As a second task, we use the most recent dataset namely *Offensive Language Identification Dataset* (OLID) [29] from the OffensEval [2] task of SemEval Workshop 2019. We act as a non-active participant for the task, and consider only the first sub-task, i.e., to classify a tweet into one of two classes - OFFENSIVE and

[1] `http://trecis.org`
[2] `https://competitions.codalab.org/competitions/20011`

NOT OFFENSIVE.

## 4.2 Dataset Statistics

### 4.2.1 TREC-IS Data

In 2018, the TREC-IS task organizers released tweets as 1335 training data. Each tweet was manually labelled with exactly one category. Due to ontology changes in the subsequent year, only 1309 samples (tweets) were retained. Test data for 2018 contained 17,682 tweets; these were used as a training data in TREC-IS v1.5 (2019)[30]. The complete data statistics are given in Table 4.1.

| **Dataset** | **#Samples** | **Label Cardinality** |
|---|---|---|
| TREC-ISv1.0 - train | 1,309 | 1.0 |
| TREC-ISv1.5 - train | 17,682 | 2.27 |
| TREC-ISv1.0 and v1.5 combined - train | 18,991 | 2.18 |
| TREC-ISv1.5 - test | 9,444 | - |

Table 4.1: TREC-IS Data Statistics

Tweets were collected from a variety of sources [25]. The combined dataset covers 21 events. The distribution of tweets across events is shown in Table 4.2

| **Events** | **#Tweets** | **Events** | **#Tweets** |
|---|---|---|---|
| joplinTornado2011 | 95 | costaRicaEarthquake2012 | 243 |
| fireColorado2012 | 259 | guatemalaEarthquake2012 | 154 |
| italyEarthquakes2012 | 103 | albertaFloods2013 | 722 |
| philipinnesFloods2012 | 437 | typhoonPablo2012 | 234 |
| australiaBushfire2013 | 677 | bostonBombings2013 | 535 |
| floodColorado2013 | 233 | laAirportShooting2013 | 160 |
| manilaFloods2013 | 411 | queenslandFloods2013 | 709 |
| typhoonYolanda2013 | 564 | westTexasExplosion2013 | 180 |
| chileEarthquake2014 | 311 | typhoonHagupit2014 | 3939 |
| nepalEarthquake2015 | 5841 | flSchoolShooting2018 | 1118 |
| parisAttacks2015 | 2066 | | |

Table 4.2: Event-wise sample distribution (TREC-IS combined)

These tweets were categorized into 24 class labels specified by the task organizers, each tweet may be assigned multiple labels. The label distribution of the dataset is given in Figure 4.1.

Among these class labels, 6 labels are identified as **Actionable** [30], since tweets

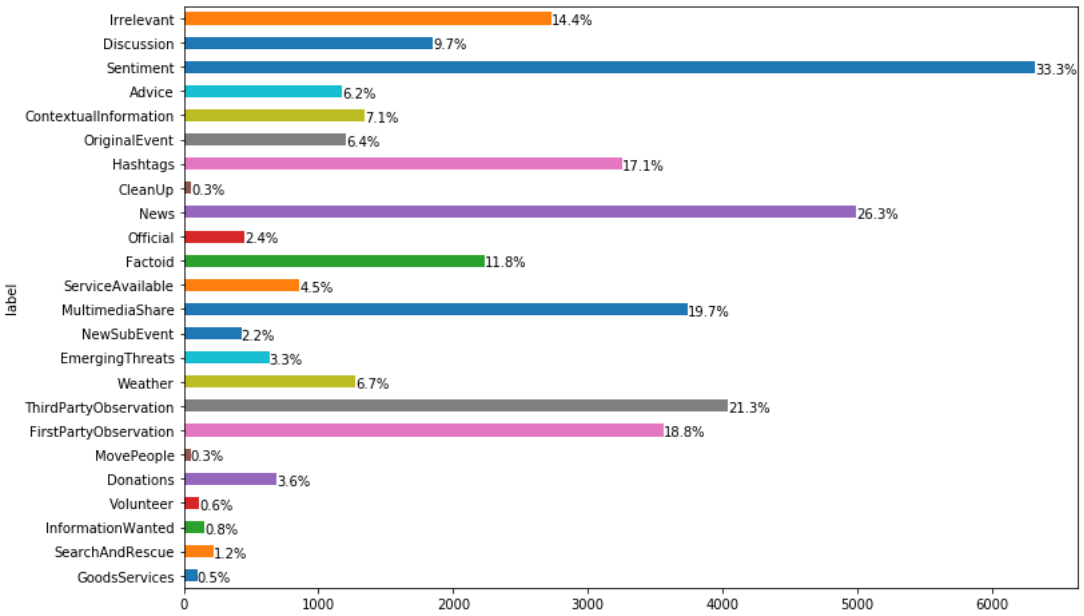



Figure 4.1: Label Distribution in TREC-IS (combined)

categorized under these labels may need a response from the management officer. These actionable labels are - 'GoodsServices', 'SearchAndRescue', 'MovePeople', 'EmergingThreats', 'NewSubEvent' and 'ServiceAvailable'.

The samples were also annotated by humans with a priority value, also referred as 'criticality' from the set {Critical, High, Medium, Low}, indicating how urgently the needs a response from the disaster management system. The distribution of priority values is shown in Figure 4.2.

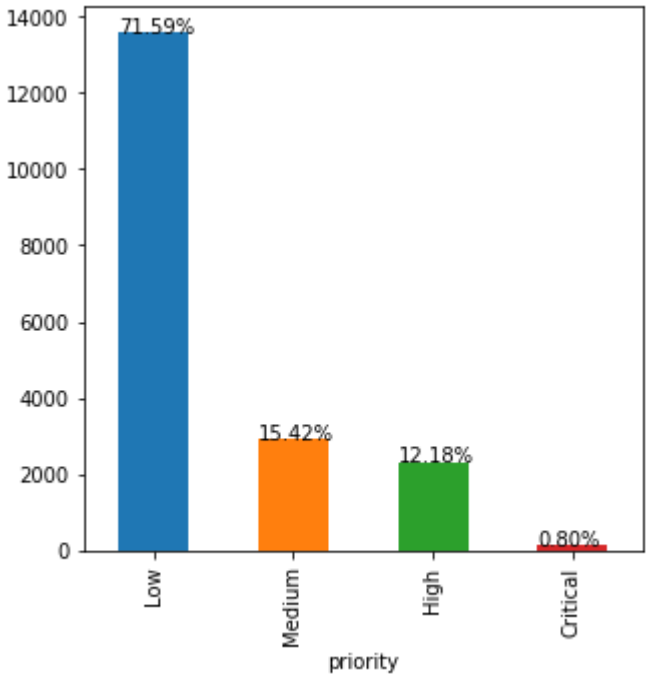



Figure 4.2: Priority Distribution in TREC-IS (combined)

We perform our experiments in a 80-20 train-test split on the TREC-IS combined dataset. Additionally, we also report the scores obtained by our official TREC-IS 2019 submissions.

### 4.2.2 OLID Data

The Offensive Language Identification Dataset (OLID) dataset by Zampieri et al. [29] contains around 14,100 tweets which is divided into training data (containing 13,240 samples) and test data (containing 860 tweets).

Each of these tweets are categorized in a hierarchical manner. For the first subtask, tweets are classified into Offensive (OFF) and Not Offensive (NOT). Offensive tweets are further classified into targeted (TIN) and untargeted (UNT). Furthermore, TIN labelled tweets were classified into individual targets (IND), Group (GRP) or other (OTH).

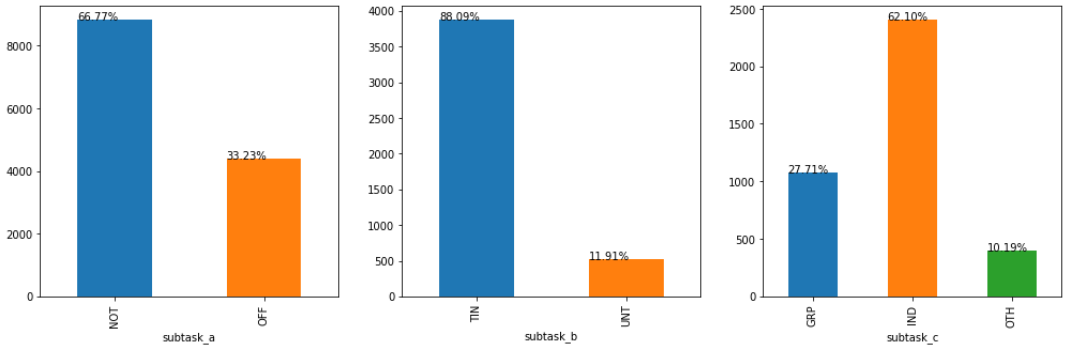



Figure 4.3: OLID Dataset distribution

In this dissertation, we only approach the first subtask, that is to classify whether a tweet is offensive or not.

## 4.3 Data Cleaning & Preprocessing

A most important part for any NLP or IR task is to clean the text data before processing. Since twitter texts are short, noisy, do not always follow any predefined format, it is difficult to work with tweet texts for any NLP/IR task.

Pre-processing steps that we used are described below,

- We used an open source text processing tool named 'ekphrasis' [31] that identifies and normalizes user mentions, hashtags, urls, numeric tokens, date-time etc.
- We used Norvig's word segmentation algorithm for segment long mis-typed tokens, hashtags (eg: *#prayforparis* → *pray for paris*), and extract smaller meaningful tokens.
- We also removed non-ASCII characters, emoji tokens, punctuation symbols etc.
- We collected a list of English contractions, and expand them in texts. (*don't* → *do not*, *should've* → *should have*).
- We also made a collection of possible social media slangs and acronyms and expand to get meaningful English dictionary words in the text.

## 4.4 Evaluation Measures

As a classification task, we enlist the usual evaluation measures used in practice, as well as the measures provided by the task organizers.
Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

$$\text{Accuracy} = \frac{\text{true positive} + \text{true negative}}{\text{true positive} + \text{false positive} + \text{true negative} + \text{false negative}}$$

Precision is the ratio of correctly predicted positive samples to the total predicted positive samples.

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

Recall is defined as the ratio of the correctly predicted positive samples to the total positive samples.

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

F1-score is the harmonic mean of Precision and Recall.

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For Multilabel classification on TREC-IS data (Ref. 4.2.1) we use the following metrics:
**Example Based Measure**

$$\text{Hamming Loss} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{L} |h(x_i) \Delta y_i|$$

where $\Delta$ stands for the symmetric difference between two sets, $N$ is the number of test examples, $L$ is the total number of class labels, $h(x_i)$ is set of the predicted labels and $y_i$ is the set of correct labels for $i$-th sample $x_i$.

**Label Based Measure**

*Macro-Precision* (Precision averaged across all labels) is defined as,

$$macro\_precision = \frac{1}{L}\sum_{j=1}^{L}\frac{tp_j}{tp_i + fp_j}$$

where $tp_j$ and $fp_j$ are number of true positives and false positives for the $j$-th label as binary class.

*Macro-Recall* (Recall averaged across all labels) is defined as,

$$macro\_recall = \frac{1}{L}\sum_{j=1}^{L}\frac{tp_j}{tp_i + fn_j}$$

where $tp_j$ and $fp_j$ are defined as the macro precision and $fn_j$ is the number of false negative for $j$-th label as binary class.

*Macro-F1* is the harmonic mean between precision and recall, where the average is calculated per label and then averaged across all labels. If $p_j$ and $r_j$ are the precision and recall for the $j$-th label, the macro-F1 is

$$macro\_F1 = \frac{1}{L}\sum_{j=1}^{L}\frac{2 \times p_j \times r_j}{p_j + r_j}$$

We enlist macro F1 score for both the problems, where for the TREC-IS task we also present Hamming Loss and for the Offensive tweets identification we present classification Accuracy. For TREC-IS task we also present the macro average F1 for actionable categories, as mentioned in task guidelines.

For the Criticality Score prediction subtask of the TREC-IS task, we follow the official metric, provided by the task organizer, calculates an Accumulated Alert Worth (AAW) described below.

The notations used in the following paragraphs are listed in Table 4.3.

| Notation | Significance |
|---|---|
| $t$ | A Tweet |
| $T$ | Set of all tweet in the Test Event |
| $T_{High/Critical}$ | Set of all annotated High or Critical tweets |
| $p_t^s$ | Priority Score predicted by the system |
| $\text{ActC}_t^s$ | Actionable Categories assigned to tweet $t$ by the system |
| $\text{ActC}_t^a$ | Actionable Categories assigned to tweet $t$ by the Assessor |
| $\text{NActC}_t^s$ | Non-Actionable Categories assigned to tweet $t$ by the system |
| $\text{NActC}_t^a$ | Non-Actionable Categories assigned to tweet $t$ by the Assessor |
| $\lambda$ | Actionable vs Non-Actionable weighting (default value = 0.75) |
| $\alpha$ | Static value for a correct alert regardless of whether the categories are correct. Default=0.3 |
| $\delta$ | The number of false alerts since the last true alert. Each time a tweet that is not in $T_{high/critical}$ is given a $p_t^s \geq 0.7$ counted as a false alert. $\delta$ is reset to 0 each time a tweet in $T_{high/critical}$ is given a $p_t^s \geq 0.7$ by the system (a true alert). This is used to emulate user trust in the system over time. |

Table 4.3: Notation for AAW calculation

Two scores namely ActCScore($t$) and NActCScore($t$) are defined as

$$\text{ActCScore}(t) = \gamma.\frac{|\text{ActC}_t^s \cap \text{ActC}_t^a|}{|\text{ActC}_t^s \cup \text{ActC}_t^a|}$$

$$\text{NActCScore}(t) = (1-\gamma).\frac{|\text{NActC}_t^s \cap \text{NActC}_t^a|}{|\text{NActC}_t^s \cup \text{NActC}_t^a|}$$

$$\gamma = \begin{cases} \lambda & \text{if } |\text{ActC}_t^a| > 0 \\ 0 & \text{otherwise} \end{cases}$$

**Scoring Tweets that should generate an alert**

For the tweets that are assessed as High or Critical category, the worth is calculated as,

$$\text{highPriorityWorth}(t) = \begin{cases} \alpha + (1-\alpha) * (\text{ActCScore}(t) + \text{NActCScore}(t)) & \text{if } p_t^s \geq 0.7 \\ -1 & \text{otherwise} \end{cases}$$

### Scoring Tweets that should not generate an alert

For the tweets that are assessed as Low or Medium category, the worth is calculated as,

$$\text{lowPriorityWorth}(t) = \begin{cases} \max(-\log(\frac{\delta}{2}+1), -1) & \text{if } p_t^s \geq 0.7 \\ \text{ActCScore}(t) + \text{NActCScore}(t) & \text{otherwise} \end{cases}$$

### Accumulated Alert Worth (AAW)

Accumulated Alert Worth (AAW) is calculated as,

$$AAW = \frac{1}{2} \begin{cases} \frac{1}{|T_{high/critical}|}.\text{highPriorityWorth}(t) & \text{if } t \in T_{high/critical} \\ \frac{1}{|T_{low/medium}|}.\text{lowPriorityWorth}(t) & \text{otherwise} \end{cases}$$

# Chapter 5

# Tweet Classification for Crisis Management System

In this chapter we discuss the first problem we approached, tweet classification for crisis scenarios. We discuss the baselines first, then the approaches we applied and then present the experimental results.

## 5.1 Approaches

### Experimental Setup

For our experiments, we consider two ways to split the data into training and testing parts. In the first approach, we split the training data into train/test (80/20) partitions for our experiment, maintaining the balance (80-20) for each class. In the second one, we mimic the underlying application scenario of the problem, by taking tweets for the older events, as mentioned in Table 4.2 as training, and tweets for the 3 latest events in the testing partition (namely 'nepalEarthquake2015', 'flSchoolShooting2018', 'parisAttacks2015').

### 5.1.1 Features

We used tf-idf feature vectorization over the cleaned and pre-processed text as mentioned in Section 4.3. Tweets were indexed with words and word bigrams.

We do not remove standard English stop-words, instead we limit our vocabulary by removing the words and word bigrams having document frequency more than 90%. This may be viewed as a form of corpus specific stop word removal. We experimented with adding additional numerical features from text such as, #words, #characters, upper case character ratio, sentiment score[32] etc and a few others such as, #user mentioned, #hashtags, #retweet count etc from tweet statistics.

### 5.1.2 Classifiers

We experimented with three broad approaches: (1) Problem Transformation for multi-label learning, (2) Algorithm Adaptation and (3) Deep Architecture based approaches. We applied Binary Relevance (BR), Classifier Chain (CC), Pruned Set (PS), RAkELd (Ref. Section 2.2.1.1) using SVM and Logistic Regression as the base classifier. We also present experiments with some algorithm adaptation methods such as, ML-kNN, PCT, RF-PCT. We experimented with some deep learning architectures such as LSTM, CNN, Bidirectional LSTM. For LSTM, BiLSTM architectures, we use a 200 dimensional hidden layer, followed by a 100 dimensional fully connected layer. Lastly we use another fully connected layer to a 24 (number of labels) dimensional output layer, and minimize the binary cross-entropy loss to train the network weights. For CNN architecture, we use 128 filters of sizes 3, 4 and 5 as suggested in [9], followed by a fully connected layer. We train the network for 20 epochs (LSTM and other variations) and 50 epochs (CNN) with learning rate 0.001 and 0.0001 respectively with batch size 16. We use both Glove [33] 200 dimensional word embedding, pre-trained on twitter as well as crisis word vectors, pre-trained on crisis related tweets by Imran et. al. [25], available publicly [1].

In order to improve the target metric *macro f-1*, rather than using hard threshold 0 for a linear classifier $f(\mathbf{x}) > 0$, we follow a threshold selection method. Since macro f-1 for a label is independent of another for the calculation of total macro f-1, the decision boundary for each classifier for a binary method can be tuned independently using different threshold selection for different label. We determine threshold for each class label using an algorithm, mentioned in [34].

[1] `https://crisisnlp.qcri.org/lrec2016/lrec2016.html`

### 5.1.3 DAG Based Label ordering for Classifier Chain (DBLCC)

In order to exploit label dependency, and improve classifier chain performance, we devise an algorithm (**Algorithm 1**) for finding an order for classifier chain method. The basic idea behind this algorithm is as follows.

**Algorithm 1** DAG Based Label ordering for Classifier Chain(DBLCC)

1: Estimate $p(y_i|y_j)$ for all $1 \leq i \leq L, 1 \leq j \leq L$ and $i \neq j$ as

$$p(y_i|y_j) = \frac{Count(y_i == 1 \wedge y_j == 1)}{Count(y_j == 1)}$$

2: $P \leftarrow$ Sorted pairs $(y_i, y_j)$ in non-increasing order of $p(y_i|y_j)$.
3: Make a Graph $G(V, E)$ where $V = \{y_1, \ldots, y_L\}$
4: **for all** $(y_i, y_j) \in P$ **do**
5:  **if** $(y_i, y_j) \notin E$ and $p(y_i|y_j) > threshold$ **then**
6:  Add a directed edge from $y_j$ to $y_i$
7:  **end if**
8: **end for**
9: Find all connected components of $G$
10: **for all** $G'$ component of $G$ **do**
11:  **if** $|G'.V| == 1$ **then**
12:  Build a one-vs-rest classifier for $y_k \in G'.V$
13:  **else**
14:  $SV \leftarrow$ list of vertices (labels) obtained by topological sort on $G'$.
15:  Build a classifier chain in order with $SV$.
16:  **end if**
17: **end for**

We build a DAG (Directed Acyclic Graph) with all class labels as vertices, and an edge according to the class conditional probability $p(y_i|y_j)$ if above a threshold. For each connected component, if there is only one vertex in that component, we build a one-vs-rest classifier for that class label, otherwise we perform topological sort on the vertices of a component. Intuitively, from our graph construction mechanism, if $y_i$ depends on $y_j$ and $y_j$ depends on $y_k$, they belong to same component with edges $y_k \rightarrow y_j \rightarrow y_i$, in this order we build a chain of classifier. At prediction time, a test sample is predicted by $k$ (number of

connected components) classifiers.

We used scikit-learn and scikit-multilearn packages for experiments on BR, CC, MLkNN, RAkELd. We used MEKA [1] library for PS with SVM as base classifier. Clus [2] was used to perform the experiments on PCT and RF-PCT. For scalability issues, the feature dimension was reduced to 5000 for the experiments on MEKA. We used PyTorch [3] module for implementing the deep learning architectures.

## 5.2 Experimental Results & Analysis

Results of our approaches are presented on Table 5.1. Binary Relevance with Logistic Regression (BR-LR) as a base classifier with bigram text features and some numerical features performs the best in terms of macro f-1, resulting 0.512 in our validation split. As suggested in [34] threshold tuning was supposed to perform better, but it does not improve the macro-F1. The reason behind this may be, as the final threshold for a label is determined by average of thresholds obtained in each fold of 5 fold of cross validation and the threshold for each fold is not so close to each other, the macro F1 from average threshold performs worse than that of the hard threshold 0. Classifier Chain (CC) performs somehow closer to the best result giving a macro-F1 of 0.497.

The other multi-label learning algorithms, that exploit label dependencies do not perform better than the simple binary relevance suggesting that there is not much significant label dependency to be inferred from the data. Whereas as suggested in [16], RF-PCT is one of the best models as per their experiments, which performs best in terms of macro-F1 for only actionable information types.

On the other hand, our proposed method is basically an ensemble of binary relevance (BR) and classifier chain (CC) guided by a label dependencies perform almost similar to the best model. The threshold mentioned in line 5 of **Algorithm 1**, acts as a choice between BR and CC. Too high a threshold results in no edges in the DAG thus a BR model is obtained, on the other hand too low

[1] `http://meka.sourceforge.net`

[2] `http://clus.sourceforge.net`

[3] `https://pytorch.org`

a threshold implies a complete CC. With our experiments, we set the threshold value with 0.45. The label ordering obtained from a topological sort of the constructed graph G, intuitively, follows the class conditional probability.

To simulate real life scenario during crisis, we have trained the same classifiers on

| **Classifier(Features)** | **Hamming Loss** | **Macro F-1 (Actionable Information type)** | **Macro F-1 (All labels)** |
|---|---|---|---|
| BR-SVM$_{linear}$ (Unigram) | 0.104 | 0.360 | 0.463 |
| BR-LR (Unigram) | 0.100 | 0.350 | 0.462 |
| BR-SVM$_{linear}$ (Bigram) | 0.079 | 0.402 | 0.481 |
| BR-LR (Bigram) | 0.080 | 0.421 | 0.504 |
| BR-SVM$_{linear}$ (Bigram + Features) | 0.081 | 0.424 | 0.489 |
| **BR-LR (Bigram + Features)** | 0.080 | 0.439 | **0.512** |
| CC-SVM$_{linear}$ (Bigram + Features) | 0.081 | 0.425 | 0.482 |
| CC-LR (Bigram + Features) | 0.082 | 0.429 | 0.497 |
| PS-SVM$_{poly}$ (Bigram + Features) | 0.079 | 0.380 | 0.414 |
| RAkELd-LR (Bigram + Features) | 0.074 | 0.425 | 0.496 |
| MLkNN (Bigram + Features) | 0.090 | 0.220 | 0.283 |
| PCT (Bigram + Features) | 0.104 | 0.501 | 0.321 |
| RF-PCT (Bigram + Features) | 0.073 | **0.526** | 0.345 |
| BR-LR + Threshold Tuning | 0.088 | 0.432 | 0.509 |
| DBLCC | 0.080 | 0.438 | 0.511 |
| LSTM + Glove | 0.084 | 0.310 | 0.391 |
| CNN + Glove | **0.072** | 0.300 | 0.402 |
| BiLSTM + Glove | 0.082 | 0.282 | 0.406 |
| BiLSTM + Attention + Glove | 0.081 | 0.368 | 0.418 |
| LSTM + Crisis-WV | 0.088 | 0.318 | 0.380 |
| CNN + Crisis-WV | 0.073 | 0.345 | 0.412 |
| BiLSTM + Crisis-WV | 0.085 | 0.268 | 0.368 |
| BiLSTM + Attention + Crisis-WV | 0.082 | 0.367 | 0.420 |

Table 5.1: Result of Classifier(s) on TREC-IS (combined) (Train-Test Split)

a separate split, based on the events, on the TREC-IS (combined) data. Considering different events at different time and location, tweet texts are more likely to belong to different vocabularies. Hence limiting the vocabulary size and thus the dimension of the feature vector to select only the top scored (tf-idf) tokens, gives

a good performance reducing the model complexity. The results of the classifiers are presented in Table 5.2. This experiment also suggests that Binary Relevance with Logistic Regression is the best model among all the classifiers.

| **Classifier(Features)** | **Hamming Loss** | **Macro F-1 (Actionable Information type)** | **Macro F-1 (All labels)** |
|---|---|---|---|
| BR-SVM$_{linear}$ (Unigram) | 0.115 | 0.049 | 0.165 |
| BR-LR (Unigram) | 0.111 | 0.069 | 0.166 |
| BR-SVM$_{linear}$ (Bigram) | 0.117 | 0.081 | 0.180 |
| BR-LR (Bigram) | 0.113 | 0.093 | 0.183 |
| *BR-SVM$_{linear}$ (Bigram + Features) | DNF | DNF | DNF |
| BR-LR (Bigram + Features) | 0.108 | 0.093 | **0.192** |
| CC-SVM$_{linear}$ (Bigram + Features) | 0.114 | 0.068 | 0.174 |
| CC-LR (Bigram + Features) | 0.112 | 0.097 | 0.191 |
| PS-SVM$_{poly}$ (Bigram + Features) | 0.105 | 0.027 | 0.125 |
| RAkELd-LR (Bigram + Features) | 0.095 | 0.057 | 0.167 |
| MLkNN (Bigram + Features) | 0.099 | 0.000 | 0.053 |
| PCT (Bigram + Features) | 0.123 | **0.256** | 0.104 |
| RF-PCT (Bigram + Features) | 0.090 | 0.164 | 0.067 |
| DBLCC | 0.107 | 0.090 | 0.191 |
| LSTM + Glove | 0.095 | 0.000 | 0.077 |
| CNN + Glove | **0.088** | 0.010 | 0.115 |
| BiLSTM + Glove | 0.095 | 0.006 | 0.084 |
| BiLSTM+ Attn + Glove | 0.094 | 0.001 | 0.095 |
| LSTM + Crisis-WV | 0.097 | 0.011 | 0.084 |
| CNN + Crisis-WV | 0.089 | 0.009 | 0.113 |
| BiLSTM + Crisis-WV | 0.093 | 0.015 | 0.082 |
| BiLSTM+ Attn + Crisis-WV | 0.096 | 0.004 | 0.093 |

Table 5.2: Result of Classifier(s) on TREC-IS (combined) (Event-wise split). *DNF signifies the Did Not Finish

From our experiments with deep learning techniques, using different word vectors do not exhibit much significant difference. Using attention mechanism along with the LSTM improves the performance. However, these methods do not perform in macro-F1 as well as the statistical models described before. In both the approaches, CNN based text classification achieves the lowest hamming loss,

although the F1-score is significantly lower than that of statistical methods.

For the other subtask, i.e., criticality score prediction, we formulate the problem as a binary classification problem by converting priority level 'High' and 'Critical' to 'label 1' and others to 'label 0'. As the evaluation measure assumes highly important tweets to have a score greater than 0.7, we scale the class prediction probability 0.5-1 to 0.7-1 and scale 0-0.5 to 0-0.69 according to the semantics of the evaluation measure described in Section 4.4. We use text features (tfidf vector of bigrams), POS tags of the tokens and the numerical features mentioned earlier. We present the performance of this classification with variety of classifiers in Table 5.3. We also present the score under the metric proposed by TREC guidelines, where the class prediction is used from the best performing model BR-LR. (Ref Table 5.1)

| **Measure** | **Classifier** | | | | | |
|---|---|---|---|---|---|---|
| | LR | SVM | Decision Tree | Random Forest | BiLSTM + Attention | CNN |
| Accuracy | 0.853 | 0.863 | 0.622 | 0.874 | 0.852 | **0.882** |
| Precision | 0.455 | 0.479 | 0.219 | **0.698** | 0.449 | 0.558 |
| Recall | 0.625 | 0.552 | **0.737** | 0.074 | 0.429 | 0.459 |
| F1-score | **0.526** | 0.513 | 0.338 | 0.134 | 0.439 | 0.504 |
| High Priority Worth | 0.035 | -0.352 | **0.185** | -0.864 | -0.246 | -0.230 |
| Low Priority Worth | 0.397 | 0.444 | 0.132 | **0.461** | 0.412 | 0.438 |
| AAW | **0.216** | 0.045 | 0.158 | -0.201 | 0.082 | 0.104 |
| RMSE | 0.259 | 0.204 | 0.334 | 0.173 | 0.322 | 0.308 |

Table 5.3: Priority Prediction for TREC-IS (combined) (Train-Test split)

As shown in Table 5.3 AAW score of a classifier built on logistic regression is higher than the other classifier. Whereas considering only high priority worth, we see Decision Tree performs the best due to high recall. Must be mentioned here, as the penalty for consecutive false positive predictions is lesser than the same for false negatives in the worth calculation (Ref 4.4), predicting almost all positive i.e., high recall increases the high priority worth. LSTM however could not perform as good as Logistic Regression and SVM whereas CNN having the highest accuracy of classification and precision better than LR and SVM.

Overall, Logistic Regression performs best with F1-score of 0.526, giving highest Accumulating Alert Worth (AAW) value 0.216 on the test split.

On submission to the TREC-IS task, our best model, BR-LR performs best among 34 global submissions. The model scores a macro averaged F1 of **0.1969** for actionable labels with a significant margin from the second best score of 0.1556. On macro F1, average across all the labels, it also gains highest score of **0.2512** whereas the second best score is 0.2312. We also gain the first position in the second subtask with the use of Logistic Regression based classification, obtaining AAW **-0.1839** in test data, where the second best AAW is -0.1973.

## 5.3 Binary Relevance : Analysis

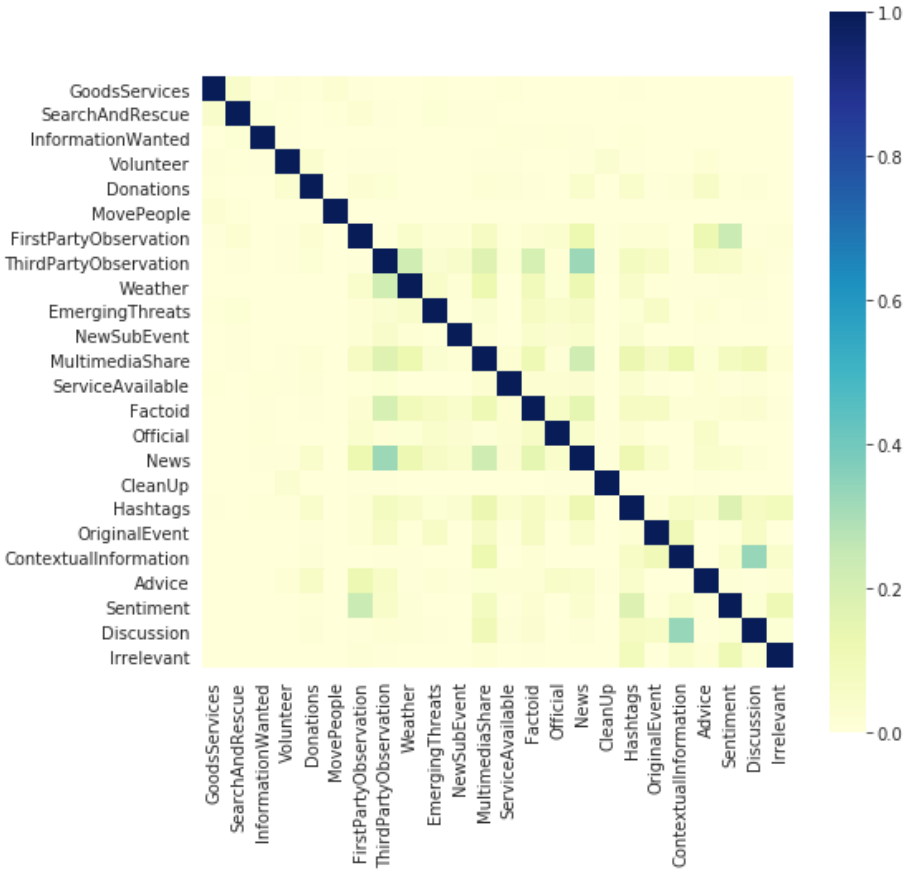



Figure 5.1: Label Co-occurence of TREC-IS (combined) data

Experiment and as well as the result on the test data suggesting Binary Relevance to be apparently the best among all the approaches, indicates there might not be any significant dependency among labels (Ref. Figure 5.1). So, in this subsection we analyze the performance of binary relevance method with the availability of the ground truth of the test data. Another shortcoming of binary relevance is independent training of classifiers for each label may produce complete class failure when for a particular test sample all the classifier fails to identify the positive class, finally predicting an empty label set for that sample.

In this scenario, one obvious choice, according to *T-Criterion* method [35] is to predict the class label with least negative output. In other words, the label for which the test sample is closest to the class boundary should be the prediction with the assumption that the classifier have just missed the sample to put into the positive side (Closest Word Assumption) [36]. We experiment with predicting the default 'Irrelevant', and also a single randomly chosen label for these scenarios. The results on validation set as well as test set are described in the Table 5.4.

| **Data** | **No-Allocation** | **T-Criterion** | **'Irrelevant'** | **Random-1** |
|---|---|---|---|---|
| validation | 0.5122/0.4396 | 0.5109/0.4326 | 0.5114/0.4396 | 0.4947/0.4127 |
| test | 0.2512/0.1969 | **0.2576/0.1976** | 0.2512/0 1969 | 0.2505/0.1955 |

Table 5.4: f-1 score for different class failure choices. The values at right side of the '/' indicates the macro averaged F1 for actionable classes and the values at the left indicates the total macro averaged F1 score.

# Chapter 6

# Offensive content identification in Tweets

## Introduction

In this section we discuss about the second task, identifying offensive content in tweets. As mentioned earlier, we only address the problem of solving whether a tweet is offensive or not.

## 6.1 Approaches

As a binary classification task, we approach this problem similar to the previous one, experimenting with both machine learning and deep learning approaches. We use tf-idf vectorization on the clean and pre-processed tweets. We present the experiments on unigram, bigram tokenizations of tweets. We also use additional text features as mention in Section 3.1. From a thorough observation, it is evident that most of the profane words are intentionally covered with characters like '*', '$' etc on Twitter and Facebook. We have also made a spell correction dictionary from such lexicons to normalize almost all manipulated tokens into similar token during pre-processing step.

We have prepared a baseline with SVM and Logistic Regression along with other classifiers such as decision tree, random forest. We also applied LSTM [7],

CNN [9] based text classification. we use 200 dimensional Glove word embedding [33], as well as 300 dimensional Fasttext word embedding trained on a large English Wikipedia corpus. We train the model by minimizing binary cross entropy loss function. The hyperparameters were determined by experimentation on a 10% validation set.

We use scikit-learn, Pytorch package of Python to implement all the statistical learning methods and the deep learning algorithm respectively.

## 6.2 Experimental Results

Results of the experiments are listed in Table 6.1. SVM with bigram tf-idf feature vectorization and with additional features performs best among statistical machine learning algorithms. Addition of Parts of Speech tags does not improve any performance significantly.

| Classifier (Features) | Accuracy | Macro F-1 |
|---|---|---|
| SVM (Unigram) | 0.791 | 0.743 |
| LR (Unigram) | 0.789 | 0.733 |
| Decision Tree (Unigram) | 0.718 | 0.67 |
| Random Forest (Unigram) | 0.812 | 0.708 |
| SVM (Bigram) | 0.792 | 0.736 |
| LR (Bigram) | 0.776 | 0.718 |
| Decision Tree (Bigram) | 0.763 | 0.719 |
| Random Forest (Bigram) | 0.790 | 0.656 |
| SVM (Bigram + Text Features) | 0.808 | **0.756** |
| LR (Bigram + Text Features) | 0.801 | 0.747 |
| SVM (Bigram + Text Features + POS Tags) | 0.795 | 0.744 |
| LR (Bigram + Text Features + POS Tags) | 0.803 | 0.748 |
| LSTM + Glove | 0.811 | 0.733 |
| CNN + Glove | 0.821 | 0.761 |
| BiLSTM + Glove | 0.811 | 0.732 |
| BiLSTM + Attention + Glove | 0.813 | **0.765** |
| LSTM + fasttext | 0.816 | 0.758 |
| CNN + fasttext | 0.833 | **0.784** |
| BiLSTM + fasttext | 0.823 | 0.766 |
| BiLSTM + Attention + fasttext | 0.828 | 0.771 |
| BERT | **0.859** | 0.741 |

Table 6.1: Result of Classifier(s) on *OLID* Dataset (Test Data)

Bidirectional LSTM model performs closer but not better than SVM, giving a F1 score of 0.732. While used with attention mechanism, the performance improves, giving a F1 score 0.765. Hence using attention is definitely a wise choice here. Where using Fast-text word embedding improves the overall results of all the deep learning architectures. Among them CNN performs the best with a F1-score of 0.784. In this problem, we observe a superiority of the deep learning techniques. Intuitively, these methods capture contextual features of the texts from the dense word vectors of the tokens and give a better feature representation for classification.

We also experimented with BERT (Ref 2.2.1), with default parameters such as maximum sequence length 128, learning rate $2 \times 10^{-5}$ and trained for 2 epochs. BERT performs best according to accuracy measure. However, it does not per-

form better than most of the deep learning models in terms of F1 score. Since, macro-F1 is unweighted average of F1 score for all the classes, higher accuracy does not imply higher F1 score due to class imbalance. We can conclude that proper hyperparameter tuning is required to get the best performance of BERT in sentence classification task.

# Chapter 7

# Conclusions

In this dissertation, we approach the problem of classification, restricted to noisy short social media text. We consider both binary and multi-label learning problems. The two main questions that we attempted to answer in the course of dissertation are:

- How important is pre-processing for social media text analysis?
- Do Deep Learning techniques offer significant improvements in performance over traditional learning techniques?

To answer the first question : our experiments suggest that pre-processing for social media text is indeed very important. Our experiments have shown that various pre-processing steps such as, normalizing word contractions, normalizing slangs, languages used by netizens, mis-typed and misspelled words, expanding emoticon meanings etc, yield small improvements which add up when these steps are combined.

The second question that we try to address here, has already been discussed by many researchers. The success of deep learning (DL) is usually attributed to the ability of DL techniques to perform automatic feature extraction from the input itself, deep learning methods are considered to effectively learn the features, when applied for some target tasks like classification etc; in many cases, a DL pipeline outperforms the traditional machine learning techniques that make use of handcrafted features.

Our experiments with 2 different datasets and variants of short text classification suggest that there is no hard and fast rule that DL always outperforms the statistical machine learning approaches. The outcome depends on many factors, like the amount of data available, the distribution of the data, the target task etc and mainly selection and tuning of hyper-parameters and the conclusion can be drawn after appropriate experiments using different approaches. Such empirical study continue to be important because the interpretability of Deep Learning is still a question of research, whereas the older machine learning approaches are relatively easy to interpret and comprehend. Further, some of the most effective DL techniques are still too computationally intensive to be used with commodity hardware.

# References


[1] L. Thapa, "Spatial-temporal analysis of social media data related to nepal earthquake 2015," vol. XLI-B2, 01 2016. 2

[2] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pp. 512–515, 2017. 2, 16

[3] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, pp. 1–47, Mar. 2002. 4, 5

[4] Y. Yang, "An evaluation of statistical approaches to text categorization," *Information Retrieval*, vol. 1, pp. 69–90, Apr 1999. 5

[5] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, 2014. 5

[6] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, pp. 21–27, Sept. 2006. 6

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 8, 36

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2015. 8

[9] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language*

*Processing (EMNLP)*, (Doha, Qatar), pp. 1746–1751, Association for Computational Linguistics, Oct. 2014. 9, 28, 37

[10] Y. Zhang and B. C. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *CoRR*, vol. abs/1510.03820, 2015. 9

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. 10

[12] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018. 10

[13] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. 10

[14] J. Howard and S. Ruder, "Fine-tuned language models for text classification," *CoRR*, vol. abs/1801.06146, 2018. 10

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 5998–6008, Curran Associates, Inc., 2017. 10

[16] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Deroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recogn.*, vol. 45, pp. 3084–3104, Sept. 2012. 11, 30

[17] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *IJDWM*, vol. 3, pp. 1–13, 2007. 11

[18] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," in *Machine Learning and Knowledge Discovery in Databases* (W. Buntine, M. Grobelnik, D. Mladenić, and J. Shawe-Taylor,

eds.), (Berlin, Heidelberg), pp. 254–269, Springer Berlin Heidelberg, 2009. 12

[19] K. Dembczyński, W. Cheng, and E. Hüllermeier, "Bayes optimal multilabel classification via probabilistic classifier chains," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (USA), pp. 279–286, Omnipress, 2010. 12

[20] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, (Washington, DC, USA), pp. 995–1000, IEEE Computer Society, 2008. 12

[21] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-labelsets for multilabel classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1079–1089, July 2011. 13

[22] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, pp. 2038–2048, 2007. 13

[23] D. Kocev, C. Vens, J. Struyf, and S. Džeroski, "Ensembles of multi-objective decision trees," in *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, (Berlin, Heidelberg), pp. 624–631, Springer-Verlag, 2007. 14

[24] H. Blockeel, L. D. Raedt, and J. Ramon, "Top-down induction of clustering trees," *CoRR*, vol. cs.LG/0011032, 2000. 14

[25] M. Imran, P. Mitra, and C. Castillo, "Twitter as a lifeline: Human-annotated twitter corpora for NLP of crisis-related messages," *CoRR*, vol. abs/1605.05894, 2016. 15, 19, 28

[26] I. Kwok and Y. Wang, "Locate the hate: Detecting tweets against blacks," 2013. 16

[27] R. Kumar, A. K. Ojha, S. Malmasi, and M. Zampieri, "Benchmarking aggression identification in social media," in *TRAC@COLING 2018*, 2018. 16

[28] J. R. Michael Wiegand, Melanie Siegel, "Proceedings of the germeval 2018 workshop," in *14th Conference on Natural Language Processing KONVENS 2018*, 2018. 16

[29] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar, "Predicting the Type and Target of Offensive Posts in Social Media," in *Proceedings of NAACL*, 2019. 18, 21

[30] C. B. Richard Mccreadie and I. Soboroff, "Trec incident streams: Finding actionable information on social media," in *16th International Conference on Information Systems for Crisis Response and Management*, ICSCRM'19, 2019. 19, 20

[31] C. Baziotis, N. Pelekis, and C. Doulkeridis, "Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, (Vancouver, Canada), pp. 747–754, Association for Computational Linguistics, August 2017. 22

[32] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text.," in *ICWSM* (E. Adar, P. Resnick, M. D. Choudhury, B. Hogan, and A. H. Oh, eds.), The AAAI Press, 2014. 28

[33] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. 28, 37

[34] R.-E. Fan and C.-J. Lin, "A study on threshold selection for multi-label classification," 2007. 28, 30

[35] M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng, "Binary relevance for multi-label learning: An overview," *Front. Comput. Sci.*, vol. 12, pp. 191–202, Apr. 2018. 35

[36] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," 2004. 35