Classification for Crisis-Related Tweets Leveraging Word Embeddings and Data Augmentation

Congcong Wang
School of Computer Science
University College Dublin
Dublin, Ireland
congcong.wang@ucdconnect.ie

David Lillis
School of Computer Science
University College Dublin
Dublin, Ireland
david.lillis@ucd.ie

ABSTRACT

This paper presents University College Dublin's (UCD) work at TREC 2019-B Incident Streams (IS) track. The purpose of the IS track is to find actionable messages and estimate their priority among a stream of crisis-related tweets. Based on the track's requirements, we break down the task into two sub-tasks. One is defined as a multi-label classification task that categorises upcoming tweets into different aid requests. The other is defined as a single-label classification task that estimates these tweets with four different levels of priority. For the track, we submitted four runs, each of which uses a different model for the tasks. Our baseline run trains classification models with hand-crafted features through machine learning methods, namely Logistic Regression and Naïve Bayes. Our other three runs train classification models with different deep learning methods. The deep methods include a vanilla bidirectional long short-term memory recurrent neural network (biLSTM), an adapted biLSTM, and a bi-attentive classification network (BCN) with pre-trained contextualised ELMo embedding. For all the runs, we apply different word embeddings (in-domain pre-trained, word-level pre-trained GloVe, character-level, or ELMo embeddings) and data augmentation strategies (SMOTE, loss weights, or GPT-2) to explore the influence they have on performance. Evaluation results show that our models perform better than the median for most situations.

KEYWORDS

Emergency response, text classification, deep learning

1 INTRODUCTION

According to a 2016 report [6], an average of 50,000 people worldwide die of natural disasters annually. This damage has greatly raised awareness of emergency response operators. Over the last few decades, a lot of work has been done for the purpose of emergency communication and coordination [14, 20, 22]. In the past, this kind of work went to using traditional online news wire to manage and track emergencies [14, 22]. More recently, with the mass adoption of social media, its importance in tracking emergencies has been recognised [10, 21, 26, 27]. Compared to traditional online centralised news wires, the user generated content on social media enables emergency aid requests to be responded to in near real-time fashion. It is reported that around 69% of people expect aid responses on social media[1] during a crisis. A recent study on Twitter also shows that around 10% of emergency-related tweets are actionable and around 1% are critical [16]. Despite these facts, many challenges still exist. One challenge is that messages are posted explosively on social media during crises, which makes it difficult and costly to find useful ones. Obviously, monitoring the emergency-related messages on social media manually is unrealistic. This motivates the development of mature computational techniques for automatically finding useful messages on social media during crises. However, little research has been undertaken for this specific problem to date.

The Incident Streams track (IS)[2] is a research initiative proposed as part of the Text REtrieval Conference (TREC)[3] aiming to mature social media-based emergency response technology. The IS track is a task originally proposed in 2018 with specific focus on categorising crisis-related tweets and estimating their priority. It provides a dataset that contains manually-annotated crisis-related tweets with respect to aid requests and priority levels. The aid requests refer to information types describing what type of aid a crisis-related tweet is requesting. Information types are broadly classified into two categories: actionable and non-actionable types (as presented in Table 1). Actionable types are assigned to tweets with more urgent needs during a crisis, such as SearchAndRescue, GoodsServices, etc. Non-actionable tweets tend to be less urgent including Volunteer, FirstPartyObservation, etc. In the dataset, each tweet is assigned to one or more information types (multi-label). Apart from information types, tweets are also annotated with one of four different levels of priority - critical, high, medium, or low (single-label). With the dataset, participants are asked to develop systems that are capable of categorising the information types and estimating the priority of upcoming unseen crisis-related tweets.

[1] https://mashable.com/2011/02/11/social-media-in-emergencies
[2] http://dcs.gla.ac.uk/~richardm/TREC_IS/
[3] https://trec.nist.gov/

Table 1: Actionable vs non-actionable information types

Actionable	Non-actionable
SearchAndRescue GoodsServices MovePeople EmergingThreats NewSubEvent ServiceAvailable	ContextualInformation Advice CleanUp News Discussion Donations Factoid FirstPartyObservation ThirdPartyObservation Hashtags InformationWanted OriginalEvent Official MultimediaShare Location Volunteer Sentiment Weather Irrelevant

In the IS track, there are 25 information types, of which 6 are defined as "actionable" and the remaining 19 as "non-actionable".

Based on the requirements, we break down the task into two sub-tasks: a multi-label classification task for information type categorisation and a single-label classification task for priority estimation. In 2019, the IS track was run twice, referring to 2019-A and 2019-B edition respectively. In the 2019-A edition, we submitted four runs, namely Run1, Run2, Run3 and Run4. All the runs firstly apply feature extraction by an in-domain pre-trained word embedding model [11], and then ensemble techniques to predict the information types of a given crisis-related tweet. Finally, a linear combination is used to compute the importance score of the tweet. The four runs are designed in parallel based on the principle of one-variable-test in one experiment. The difference between Run1 and Run2 is whether an actionable-specific classifier is used in the hope of improving performance in categorising actionable tweets. The difference between Run3 and Run4 is whether 21 hand-crafted features are combined with the in-domain word embedding model in the hope of boosting performance in priority estimation. The evaluation results show that these runs are better performing in classifying information types than most participant runs, although they are somewhat conservative in discovering tweets for alerting purposes. In addition, the runs have a better distribution across classes that are needed for information type categorisation. Having not participated in the 2018 edition of the IS track, we used the 2019-A edition as an initial testing ground for evaluating potential approaches to the challenge.

Inspired by our work at the 2019-A edition, we submitted four new runs for the 2019-B edition. This paper focuses primarily on this edition. Our baseline run is based on Run3 from the 2019-A edition, and applies machine learning approaches with hand-crafted features. It trains models through Logistic Regression and Naïve Bayes. Our other three runs apply deep learning methods including a vanilla biLSTM, an adapted biLSTM and a BCN+ELMo [15, 24] network respectively. The evaluation results provided by the organisers show that our runs have an advantage over most participating groups, based on the median performance metrics.

The paper is organised as follows. Section 2 gives a brief review of related work, and then Section 3 presents an overview of our system's architecture. For the implementations of each run, Section 4 describes the variations between the models that we trained for the TREC-IS task. Finally, Section 5 reports the experimental setup and evaluation results.

2 RELATED WORK

Word Embedding

Word embedding is an important component for deep learning models in natural language processing (NLP). It works by converting words in a vocabulary to vectors of real numbers with the objective of generating similar vector representations for semantically similar words [1, 17, 18, 23]. In the literature, neural networks and matrix factorisation are two common ways to learn word embeddings. Perhaps the most classic word embedding example is word2vec [17, 18]. It learns word representations by applying an efficient neural network prediction model. It has been demonstrated to be very effective for improving performance in language understanding tasks. Due to its effectiveness, word2vec has catalysed research in word embeddings and has sparked many extensions. An important follow-up work is fastText [1], which extends word2vec to enrich word vectors with subword information. Namely, it treats each word as being composed of character n-grams instead of a word whole as is done in word2vec. This feature enables fastText not only to generate better word embeddings for rare words but also for out-of-vocabulary (OOV) words. As for word embeddings through matrix factorisation, the representative work is GloVe [8, 23]. This learns efficient word representations by performing training on aggregated global word-word co-occurrence statistics from a corpus. More recently, word embeddings have been extended to context-based learning for word representations. Unlike the word embedding approaches as described, context-based approaches can generate different vectors for the same word with different contexts. In the literature, ELMo (Embeddings from Language Models) [24] is a good example

of context-based word embedding approaches. It learns contextualised word vectors based on a biLSTM language model and has achieved state-of-the-art performance in many language understanding tasks.

Text Classification

Text classification as an important aspect of NLP has been widely studied for several decades [13, 15, 28, 30]. The task describes input as a text entity (sentence/document) and output as the category that the text entity belongs to. Given text consisting of a sequence of words, the task of text classification can be transferred to a task of sequence representation learning. With word embeddings providing vector representations for the words in a sequence, one straightforward way to generate the sequence's representation is so-called bag-of-means (BOM) [12]. BOM generates a sequence representation by simply averaging the vectors of all words in the sequence. Once the sequence representation has been obtained, it can be fed as features for subsequent classifiers such as Logistic Regression or Naïve Bayes. In the literature, most work nowadays in sequence representation learning for text classification has been using deep neural networks. For example, Kim [13] applies convolutional neural networks (CNN) for sentence classification. Zhou et al. [30] combine CNN with LSTM for text classification. Yang et al. [28] apply hierarchical attention networks for document classification. Zhou et al. [31] improve text classification by integrating bidirectional LSTM with two-dimensional max pooling. More recently, Peters et al. [24] apply a bi-attentive classification network (BCN) with pre-trained contextualised ELMo embedding, achieving state-of-the-art performance in text classification.

3 SYSTEM ARCHITECTURE

Figure 1 shows the overall system structure that all our runs are based on. The process begins by preprocessing the tweets in the training set. The preprocessing mainly includes data cleaning and refining. For our baseline run, we convert all tweets to be lower-cased and removed stopwords, URLs, numbers, punctuation and special characters (like #, &, @, etc.). In addition, we borrow an in-domain, out-of-vocabulary (OOV) dictionary [11] for correcting the misspelled words in tweets. For our neural network based runs, preprocessing only involves the removal of special characters and the application of OOV corrections.

We notice after analysing the dataset that its classes were severely imbalanced, given that it contains few tweets with more urgent information types and more critical priority levels. After preprocessing, due to the imbalanced actionable classes in the training set, for some runs we apply a data augmentation strategy for alleviating this problem. Our baseline run applies the oversampling strategy SMOTE [2] for data augmentation, which statistically generates new representations from existing feature representations. The adapted biLSTM run does not apply data augmentation. Vanilla biLSTM and BCN+ELMO runs apply the state-of-the-art text generation strategy GPT-2 [25] for data augmentation. For GPT-2, we extracted rarer tweet samples with respect to information types and priority in the training set and then use the extracted samples as conditional samples for GPT-2 to generate synthetic samples. Table 2 gives an example of a generated sample by GPT-2 given a raw critical tweet from training set as the conditional sample.

Table 2: Generation of a critical tweet by GPT-2

Conditional raw sample	Generated sample
LANDSLIDE!... road blocked in Costa Rica after M7.6 earthquake	1 car submerged on a hillside and one car in water

After preprocessing and augmentation, the tweets are fed to train two models separately for information type detection and priority estimation purposes. The models trained at this step vary in our four different runs. Section 4 details the internals of the models that are trained for our runs. Once the models are trained, we use them to make predictions for the unlabeled tweets in the test set. At this stage, each test tweet is predicted with one or more information types and one priority label. As the submission requires the estimation of a tweet's priority with a score between 0 and 1, an extra step considered for better estimating the priority score is linear combination. The linear combination formula is defined as follows:

$$s_i = (1 - \lambda) \times w_i + \lambda \times f(p_i) \quad (1)$$

where, given a tweet i, s_i refers to its submitted priority score and w_i refers to the averaged priority weight. Given the predicted information types predicted for tweet i, w_i is the averaged weight by looking up an information-type-to-weight table as presented in Figure 1. The look-up table is constructed by quantitatively analysing all tweets in the training set (the weight for an information type T is calculated by averaging the priority scores of the tweets that belong to T in the training set). In the equation, f is the mapping function that maps the predicted priority label p_i to the numeric priority score according to the mapping scenario: low=0.25, medium=0.5, high=0.75 and critical=1.0. In the formula, λ is a constant set up to be 0.5 by default, to give equal contribution to the predicted priority label and analysed priority weight. Through the linear combination process, priority is quantified to a score between 0 and 1 for each test tweet.

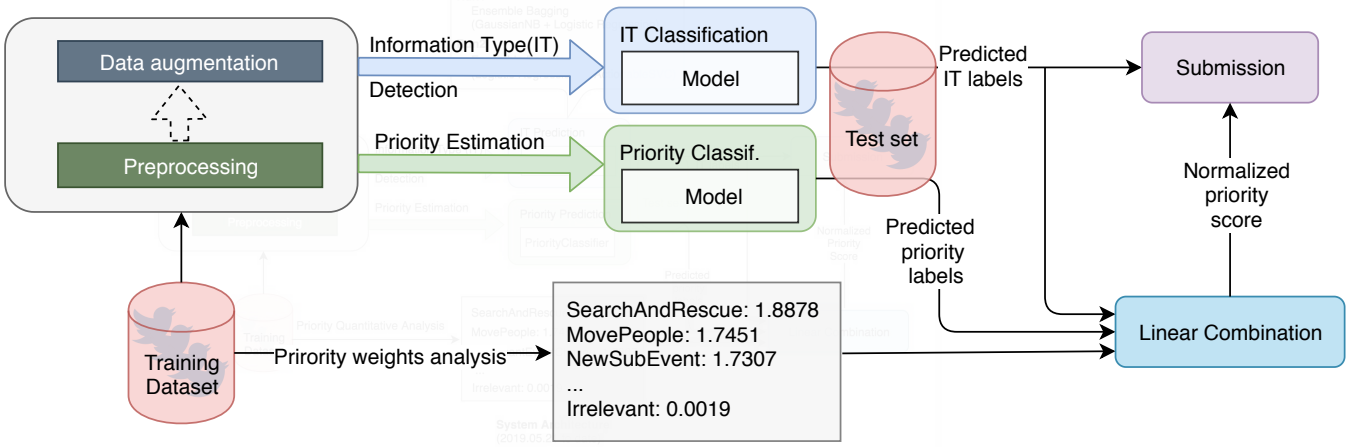

Figure 1: System architecture

4 MODELS

In this section, we describe four different model variations for our four runs, as in Table 3.

Table 3: UCD's runs with respect to model variations

Runs	Models
UCDbaseline	Logistic Regression + Naïve Bayes
UCDbilstmalpha	Vanilla biLSTM
UCDbilstmbeta	Adapted biLSTM
UCDbcnelmo	BCN+ELMo

Logistic Regression and Naïve Bayes

For our baseline, we train two models for the multi-label classification task (information type) and single-label classification task (priority) respectively. The training steps we use for model training are almost the same except for the different training objectives. First, we extract features for training tweets before being fed to train models. For feature extraction, inspired by work that has been done in the literature relevant to this problem [3, 9], eventually 21 hand-crafted features are extracted for each raw tweet. The following list summarises the 21 features:

- Number of hashtags (numeric)
- Number of "special" verbs, such as, trapped, stuck, move etc (dataset statistical analysis)
- Sentiment polarity (categorical, -1, 0 or 1)
- Tweet length (word_length, char_length, numeric)
- URL count (numeric)
- Digit count (int, numeric)
- Retweet check (0 or 1)
- Capital ratio (float, numeric)
- Special characters count (@, ! and ?, normalised float numeric)
- Named Entity count (numeric)

Apart from the hand-crafted features, we also apply a pre-trained in-domain 300-dimensional embedding [11] for word representations. To generate a tweet representation, we a apply the bag-of-means (BOM) strategy, which averages the vectors of all individual words in the tweet. Finally, the concatenation of the tweet's representation and its hand-crafted features are fit on a Logistic Regression classifier and a Naïve Bayes classifier. For priority classification, the two classifiers work together to vote for the priority label that is predicted for a tweet. For information type classification, `OneVsRest` solves the multi-label classification problem by decomposing it into multiple independent binary classification problems. The information types predicted for a tweet are also decided by the two classifiers with a combined probability threshold=0.51 to check whether an information type should be assigned.

Vanilla biLSTM

Our second run trains models based on a vanilla biLSTM network. Although two models are trained separately for priority classification and information type classification, they follow a similar structure. The model's structure is straightforward. Given a tweet t consisting of a sequence of words:

$\langle w_0, w_1, w_2, ..., w_t, ...w_n \rangle$, we embed the words using GloVe[4] to word vectors $\langle x_0, x_1, x_2, ..., x_t, ...x_n \rangle$. Next, we use the standard LSTM [7, 31] to encode the word vectors to a single sequence representation. The standard LSTM is a memory-based unit in a recurrent neural network (RNN). It helps to summarise the state of the previous t tokens in a sequence given a token vector x_t at time step t. The LSTM unit takes x_t with previous hidden state h_{t-1} as input to output a new hidden state h_t that summarises important information of the sequence so far (from time step 0 to t). It summarises important information by remembering or forgetting information through a set of transition functions (or so-called gates) in the unit. The LSTM transition functions are defined as follows [31]:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}; x_t] + b_i) \\ f_t &= \sigma(W_f \cdot [h_{t-1}; x_t] + b_f) \\ q_t &= tanh(W_q \cdot [h_{t-1}; x_t] + b_q) \\ o_t &= \sigma(W_o \cdot [h_{t-1}; x_t] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot q_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{2}$$

where σ and $tanh$ stand for sigmoid and hyperbolic tangent activation functions respectively. Element-wise multiplication is denoted by $\odot$ and ; refers to the concatenation of vectors. To better understand how the transition works, we can describe f_t as the forget gate that controls what information from the old memory cell should be forgotten. We can also describe i_t as an input gate to control what new information is to be stored in the current memory cell and o_t as an output gate to control what to output based on the memory cell c_t. In LSTM-based RNNs, the above process happens concurrently from time 0 to n. Hence, a LSTM encoder eventually outputs a single vector h_n at the last time step that summarises a sequence.

Unlike a LSTM, which summarises information in one direction from left to right, biLSTM sees the context of a token at time step t in both left-to-right and right-to-left directions. In biLSTM, the transition functions in its left-to-right process are the same as described in Equation 2. The only difference in the right-to-left process is that h_t is generated based on the next hidden state h_{t+1} and the token vector x_t at time step t. To make the difference, we denote $h_n^{\rightarrow}$ as the last state in left-to-right process and $h_0^{\leftarrow}$ as the last state in right-to-left process. We concatenate the two states to a single high-level representation h_s for the sequence and then pass it through a feed-forward network.

In the feed-forward network, the output layer is different for the information type categorisation and priority estimation tasks. Because priority estimation is defined as a single label classification problem, cross entropy loss with soft-max over all classes is selected for the objective function. For the multi-label information type classification, we adapt the binary cross entropy loss [19] as the objective function that is given in Equation 3. For a training tweet, the loss is calculated between its feed-forward output logits x and its target information types y (i.e. ground-truth), as follows:

$$loss(x, y) = -\sum_{l \in L} w_l \cdot [y_l \cdot \sigma(x_l) + (1 - y_l) \cdot (1 - \sigma(x_l))] \tag{3}$$

where σ is the logistic sigmoid function and L denotes the total number of information types. In the equation, w_l denotes loss weight that is leveraged on the current label l for alleviating the label's imbalanced proportion in the training set. This loss weight is adjustable. In the vanilla biLSTM run, due to the use of GPT-2 for data augmentation, we do not put the loss weight in effect and thus is set to 1 for every label. The loss weight is included so that it can be used by later runs that do not include a data augmentation step.

Adapted biLSTM

Our third run trains models with two main differences from Vanilla biLSTM. Firstly, GPT-2 is applied for data augmentation in the Vanilla biLSTM run. Considering that text generation by GPT-2 may introduce noisy data in the training set, we do not apply GTP-2 for data augmentation in the adapted biLSTM run. Instead, to address the imbalanced class problem, we apply loss weights in objective functions of both cross entropy and binary cross entropy. As in Equation 3, w_l is set up according to the imbalanced proportions of classes instead of 1 as in the Vanilla biLSTM run. The leveraging of loss weights enables the training set to appear to be balanced for the objective functions.

Secondly, inspired by the work in [1, 29], for word embedding we not only keep the GloVe embedding but also introduce a character-level embedding. The character-level embedding is generated through a CNN that encodes n-gram characters of a word to a single representation. The character-level embedding has an advantage over word-level embedding in that it can interpret misspelled words or rare words in the vocabulary. This is because two words are converted to similar vectors through this kind of embedding as long as they are similar in morphology. In this run, given the context of tweets that often contain misspellings or abbreviations, we combine the two kinds of embedding with the aim of improving classification performance.

BCN+ELMo

In our last run, we use a neural network architecture that has been recently shown to perform well on text classification tasks [15, 24]. Bi-attentive classification (BCN) was first

[4]https://nlp.stanford.edu/projects/glove/

introduced in [15]. BCN takes two sequences as input and applies a biLSTM encoder to create task-specific representations of each input sequence. One noticeable feature of the network is the application of a biattention mechanism. The attention mechanism is used to pay attention of each input representation to the other (so that one is conditional on the other). After the biattention process, conditional information of one on the other is obtained. BCN then again applies a biLSTM to integrate the conditional information before going to the output layer. In the output layer of BCN, a max-out network uses pooled features to compute a distribution over possible classes. Peters et al. [24] combined BCN with ELMo to achieve state-of-the-art performance in sentiment classification. BCN+ELMo applies the contextualised ELMo word representations in the embedding layer of BCN. Based on the promising performance that BCN+ELMo achieved in similar tasks, we adapt it to our problem domain as the last run of our submissions.

In the embedding layer, we not only use ELMo but also combine it with GloVe. In the priority classification task, the objective function uses cross entropy loss as in the original BCN+ELMo. For the information type classification task, we still use the function in Equation 3 as the objective function.

5 EXPERIMENTAL SETUP AND EVALUATION

Based on the four model variations described in Section 4, we run four experiments for training. As in Table 3, the four experimental runs are UCDbaseline, UCDbilstmalpha, UCDbilstmbeta and UCDbcnelmo. In UCDbaseline, the C parameter of Logistic Regression is set to 1 empirically and other parameters were given their default values as in *scikit-learn*[5].

For our three neural network based runs, the common hyperparameters are batch size = 64, number of epochs = 40 with patience = 5 for early stopping, adam as optimizer with learning rate=0.001. In particular, we empirically set a threshold 0.1 to decide if an information type is assigned to a tweet in the multi-label information type classification task. Experiments on these runs are supported by *AllenNLP* [5].

In both the UCDbilstmalpha and UCDbilstmbeta runs, pre-trained `glove.6B.100d` is set to be trainable in the embedding layer. In UCDbcnelmo, pre-trained `glove.840B.300d` combined with pre-trained original size ElMo are set to be trainable in the embedding layer. Regarding data augmentation, SMOTE is available in the *scikit-learn* library and aGPT-2 implementation is included in the *transformers* library[6]. The code for reproducing our runs is open sourced at Github[7].

[5]https://scikit-learn.org/

[6]https://github.com/huggingface/transformers

[7]https://github.com/wangcongcong123/UCDTrecIS2019

With all necessary experiments set up, we make predictions on the tweets in the test set with the trained models. We finally submit the results of the four runs to the TREC organisers for evaluation. The evaluation process is straightforward. All participant groups submit their runs to TREC, and the track organisers employ human assessors to label all of the submitted tweets. Then, the evaluation results are returned to the participant groups so that they know the overall performance of their runs among different groups. Here we report UCD's results at TREC IS 2019-B track.

Table 4 presents the returned evaluation results for UCD's four runs. As can be seen from this table, there are three main categories of evaluation metrics. The first category is "Alerting", whose accumulated alert worth (AAW, ranges from -1 to 1) is used to indicate how effectively a system identifies critical and actionable tweets for alerting. AAW also applies a penalty on a system that pushes too many false alerts even it is able to find many actionable tweets. AAW emphasises the importance of information type prioritisation. The second category "Information Feed" (ranges from 0 to 1) includes two main metrics: information type F1 and accuracy. Both are used to test how accurately a system classifies information types, The difference in performance between categorising actionable tweets and non-actionable tweets is important so actionable F1 and All F1 are both included. This evaluation category emphasises the performance of a system in information type categorisation. The last category "Prioritisation" applies root mean squared error (RMSE) to test how well a system estimates the importance score of tweets in terms of priority. For the three categories of metrics, only RMSE is the case that lower is the better (ranges from 0 to 1 because it is based on underlying prioritisation scores that themselves range from 0 to 1).

Apart from the scores that our four runs achieved in Table 4, the median scores for each metric are also included. Overall, our runs outperform the median for most situations. First, UCDbaseline achieves good performance in prioritisation. Its AAW score is also slightly above the median but its information type accuracy is somewhat distant from the median. We conclude that UCDbaseline has the ability to find actionable tweets but in a way that assigns too many information types to tweets.

Comparatively, UCDbilstmalpha is somewhat weak among the four runs. It hits an information type accuracy 0.86, which is marginally above the median of 0.8583. However, it does not do well in AAW, which indicates its weakness in correctly finding tweets for alerting. When it comes to UCDbilstmbeta, we see some improvements over UCDbilstmalpha. In particular, UCDbilstmbeta achieves good performance in AAW, whose scores (-0.6047 and -0.3332) are increased from the median (-0.9197 and -0.4609) to some extent. It also achieves better score in information type positive actionable

Table 4: Evaluation results of UCD's four runs at TREC 2019-B Incident Streams track

	Alerting		Information Feed			Prioritisation	
	Accumulated Alert Worth		Info. Type Postive F1		Info. Type Accuracy	Priority RMSE	
Runs	High Priority	All	Actionable	All	All	Actionable	All
Median	-0.9197	-0.4609	0.0386	0.1055	0.8583	0.1767	0.1028
UCDbaseline	-0.7856	-0.4131	**0.1355**	**0.1343**	0.7495	**0.0859**	**0.0668**
UCDbilstmalpha	-0.9287	-0.4677	0.0614	0.1087	**0.86**	0.1521	0.0893
UCDbilstmbeta	**-0.6047**	**-0.3332**	0.1269	0.0607	0.8378	0.1004	0.0822
UCDbcnelmo	-0.6961	-0.3624	0.1099	0.1192	0.8452	0.1036	0.0769

F1 than UCDbilstmalpha. Its performance over UCDbilstmalpha implies that the use of character-level embedding and loss weights in UCDbilstmbeta helps improve the overall performance. Compared to the other three runs, it is shown that UCDbcnelmo obtains relatively even good performance across the metrics. It achieves better scores in almost every metric than the median to a good degree except that its information type accuracy is slightly lower than the median. We conclude that the BCN+ELMo achieves competitive performance in the IS track problem domain.

6 CONCLUSION AND FUTURE WORK

We have described UCD's participation at TREC 2019-B IS track for which we submitted four runs. This paper presents the techniques we used for the four runs and reports evaluation results of the runs. Our baseline run trains models through machine learning methods including Logistic Regression and Naïve Bayes. Our focus on this run is to extract 21 hand-crafted features for model training. We also leverage a pre-trained in-domain embedding combined with the 21 features for performance boosting. In this run, SMOTE is used for data augmentation due to the imbalanced classes. The evaluation shows it is able to find many actionable tweets but in a way that assigns too many information types to tweets.

Our other three runs train models through neural network based methods. In these runs, we introduced GPT-2 or loss weights for alleviating the imbalanced class problem in the training set. We also apply the commonly-practised GloVe or contextualised ELMo embedding to enrich representations for the three runs. The evaluation shows that UCDbilstmalpha is somewhat conservative in finding tweets for alerting, UCDbcnelmo achieves improvements over UCDbilstmalpha likely due to the use of character-level embedding and loss weights, and UCDbcnelmo obtains relatively even good performance across the metrics.

For future work, we expect to balance the trade-off between accuracy and other metrics for UCDbaseline. We aim to extract more informative features so as to improve its overall performance. In addition, regarding deep learning methods for the track, we have not yet explored whether applying recent advances in pre-trained language models like BERT [4] for fine-tuning on classification tasks could add further benefits. This is certainly another avenue that we will seek to explore in the future.

REFERENCES

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information, Vol. 5. 135–146. https://doi.org/10.1162/tacl_a_00051

[2] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2019. SMOTE: Synthetic Minority Over-Sampling Technique, Vol. 16. 321–357. http://dl.acm.org/citation.cfm?id=1622407.1622416

[3] Stefano Cresci, Maurizio Tesconi, Andrea Cimino, and Felice Dell'Orletta. 2015. A linguistically-driven approach to cross-event damage assessment of natural disasters from social media messages. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 1195–1200.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[5] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. Association for Computational Linguistics, 1–6. https://doi.org/10.18653/v1/W18-2501

[6] Debarati Guha-Sapir, Femke Vos, Regina Below, and Sylvain Ponserre. 2011. Annual disaster statistical review. *Centre for Research on the Epidemiology of Disasters*.

[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 1735–1780.

[8] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 873–882.

[9] Muhammad Imran, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. 2015. Processing social media messages in mass emergency: A survey. *ACM Computing Surveys (CSUR)* 47, 4, 67.

[10] Muhammad Imran, Carlos Castillo, Ji Lucas, Patrick Meier, and Sarah Vieweg. 2014. AIDR: Artificial intelligence for disaster response. In

Proceedings of the 23rd International Conference on World Wide Web. ACM, 159–162.

[11] Muhammad Imran, Prasenjit Mitra, and Carlos Castillo. 2016. *Twitter as a Lifeline: Human-Annotated Twitter Corpora for NLP of Crisis-Related Messages*. LREC. http://repositori.upf.edu/handle/10230/36883

[12] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, 427–431. https://www.aclweb.org/anthology/E17-2068

[13] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[14] Laura Lambert, Christos JP Moschovitis, Hilary W Poole, and Chris Woodford. 2005. *The internet: a historical encyclopedia*. Vol. 2. ABC-CLIO.

[15] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*. 6294–6305.

[16] Richard McCreadie, Cody Buntain, and Ian Soboroff. 2019. TREC Incident Streams: Finding Actionable Information on Social Media. In *16th International Conference on Information Systems for Crisis Response and Management (ISCRAM)*. 691–705.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[19] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. 2014. Large-scale multi-label text classification—revisiting neural networks. In *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 437–452.

[20] Fran H Norris. 2006. *Methods for disaster mental health research*. Guilford Press.

[21] Alexandra Olteanu, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. 2014. Crisislex: A lexicon for collecting and filtering microblogged communications in crises. In *Eighth International AAAI Conference on Weblogs and Social Media*.

[22] Leysia Palen and Sophia B Liu. 2007. Citizen communications in crisis: anticipating a future of ICT-supported public participation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 727–736.

[23] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[24] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2227–2237. https://doi.org/10.18653/v1/N18-1202

[25] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.

[26] Aleksandra Sarcevic, Leysia Palen, Joanne White, Kate Starbird, Mossaab Bagdouri, and Kenneth Anderson. 2012. Beacons of hope in decentralized coordination: Learning from on-the-ground medical twitterers during the 2010 Haiti earthquake. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 47–56.

[27] Kate Starbird, Leysia Palen, Amanda L Hughes, and Sarah Vieweg. 2010. Chatter on the red: what hazards threat reveals about the social life of microblogged information. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 241–250.

[28] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 1480–1489. https://doi.org/10.18653/v1/N16-1174

[29] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-Level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 649–657. http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf

[30] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. 2015. A C-LSTM Neural Network for Text Classification. *ArXiv* abs/1511.08630.

[31] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text Classification Improved by Integrating Bidirectional LSTM with Two-Dimensional Max Pooling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, 3485–3495. https://www.aclweb.org/anthology/C16-1329