Comparing performance of classifiers applied to disaster detection in Twitter tweets – preliminary considerations

Maryan Plakhtiy[1], Maria Ganzha[1][0000−0001−7714−4844], and Marcin Paprzycki[2][0000−0002−8069−2152]

[1] Warsaw University of Technology, Warsaw, Poland
M.Ganzha@mini.pw.edu.pl
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
marcin.paprzycki@ibspan.waw.pl

Abstract. Nowadays, disaster "detection", based on Twitter tweets, has become an interesting research challenge. As such it has even found its way to a Kaggle competition. In this work, we explore (and compare) multiple classifiers, applied to the data set from that challenge. Moreover, we explore usefulness of different preprocessing approaches. We experimentally establish the most successful pairs, consisting of a preprocessor and a classifier. We also report on initial steps undertaken towards combining results from multiple classifiers into a meta-level one.

Keywords: Tweet analysis · text preprocessing · classifiers · performance comparison · meta-classifiers.

1 Introduction

Currently, social media like Facebook, Twitter, Instagram, and others, are one of the most followed sources of news. Almost all news agencies, organizations, political parties, etc., post their information/news/advertisements there, as they are the most visited web sites. However, each of these news outlets has its own purpose and/or target audience. For example, Facebook is known to be a political discussion arena, Instagram is used by bloggers and businesses, to advertise their products. Finally, Twitter, because of its short messages, has become one of the fastest, and widely used, news/events "spreaders". Thus it became one of the most important sources of communication in the case of an emergency. Particularly, this is the case, when almost every person has a smartphone, which lets them "immediately" announce an observed emergency. As a result, a growing number of news agencies, emergency organizations, local government branches, and others, became interested in monitoring Twitter for emergency announcements. This is to help them to act faster and, possibly, save lives.

In this context, a very interesting question arises: how to distinguish an actual emergency-announcing tweet, from a "fake news tweet", clickbait, or a non-related tweet. While relatively easy for humans, making this distinction is quite challenging for computers. Therefore, event detection, on the basis of tweets' content, is a popular research area. While the most popular seems to be the sentiment analysis, ongoing research involves also fake news detection, text classification, or disaster detection. To stimulate work, related to the latter one, a competition was created on the Kaggle site [10]. Unfortunately, before the submission deadline, a leakage of correct predictions occurred, and the competition had to be canceled. Nevertheless, this work is based on the data set originating from this competition, and its aim is to apply (and compare performance of) well-known classifiers, to analyse content of tweets, to detect disasters. Moreover, interactions between data preprocessing that can be applied to the tweets and the classifiers is investigated. Interestingly, there is not much work that would delve into this topic. Finally, preliminary results, related to combining "suggestions" from multiple classifiers, and creating a meta-classifier, are presented.

In this context, we proceed as follows. In Section 2 we summarize most pertinent related work. Next, in Section 3, we describe the Kaggle data set and present summary of basic statistical analysis that was applied to it. We follow, in Section 4, with the description of data preprocessing that have been experimented with. Moreover, in Section 5, the experimental setup is outlined. Material presented up to this moment provides the foundation and allows to report, in Section 6, the experimental results obtained when different classifiers have been paired with different preprocessors and applied to the prepared data. Finally, in Section 7, initial results concerning combining multiple classifiers are introduced.

2 Related work

Let us now summarize the state-of-the-art of classifiers that can be (and have been) used in context of disaster detection (or other forms of tweet content analysis). We focus our attention on most recent, successful approaches that may be worthy further examination.

In [13], authors summarized effective techniques for data analysis in the context of Twitter data. They also suggested that Logistic Regression [25] is best suited for the disaster management applications. Obtained results supported this claim. In a somewhat similar research, reported in [2], authors analysed damages caused by the disaster (e.g. how many people were injured, homes destroyed, etc.). They have created a Tweedr (Twitter for Disaster Response) application, which consisted of: classification, clustering, and extraction. Among used classifiers, the Logistic Regression was proven to be the most effective, with the $\sim 88\%$ accuracy for the disaster detection. However, the data set used for tests, was really small and skewed (by over abundance of "positive examples").

In [24], authors leveraged a Hybrid CNN-LSTM classifier architecture, for the Fire Burst Detection. Unfortunately, they did not provide details of achieved accuracy, only claim that this approach was successful. Hybrid CNN-LSTM was chosen, based on the [1], where LSTM, LSTM-Dropout, and Hybrid CNN-LSTM, were used for the Fake News Identification on Twitter. Here, 82.29% accuracy was achieved with the LSTM, 73.78% with the LSTM-Dropout, and 80.38%

with the Hybryd CNN-LSTM (on the PHEME data set [28]), which consisted of $\sim 5,800$ tweets. These classifiers are promising, since they do not need the context, and the considered data set involves different types of disasters.

In [26], authors proposed a Hierarchical Attention Networks (HAN), for the document classification, which outperformed other considered classifiers. They compared several popular methods, testing them on six classification tasks, and stated that HAN was the best in each case. Note that authors worked with complete documents, whereas in Twitter, an entry is limited to 140 characters. Another paper supporting use of the attention mechanism is [27], where authors address the hyperpartisan (highly polarized and extremely biased) news detection problem. Authors compared different classifiers on a data set consisting of million articles, and claimed that the Bidirectioanal LSTM model, with self attention (Bi-LSTM-ATTN), performed best. Similarly to [26], they worked with large documents. However, they reduced them to 392 tokens (during preprocessing), removing stopwords, and selecting a 40000 token vocabulary. In this setup (using Bi-LSTM-ATTN), they obtained 93,68% accuracy for the hyperpartisan news detection, while other models reached accuracy of approximately 89.77-91.74%.

One of interesting approaches, to the problem at hand, is the Bidirectional Encoder Representations from Transformers (BERT; [5, 8]), which obtained state-of-the-art results in a wide array of Natural Language Processing (NLP) tasks. In [14], BERT was used for classification of tweets related to disaster management. There, performance of BERT-default, and its modifications, was compared to the baseline classifier, when applied to the combination of CrisisLex [3] and CrisisNLP [4] data sets, consisting jointly of 75800 labeled tweets. Overall, all BERT classifier outperformed the baseline one, and BERT-default had the best recall score ($\sim 64.0\%$ accuracy).

Separately, work reported in [7, 23], where authors combine suggestions from multiple sources to deliver improved results, may be worthy considerations. In this context note that different classifiers independently determine, if a given tweet is (or is not) reporting an actual disaster. Hence the question: can suggestions from multiple classifiers be combined (e.g. using a simple neural network), resulting in a better accuracy of classification?

In the mentioned researches, data preprocessing is applied in each case, but is not comprehensively considered. Therefore, besides testing known classifiers, we have decided to study effects of preprocessing on the performance of the preprocessor+classifier pairs. This is one of important contributions of this paper. The list of preprocessing methods that have been tried, can be found in Section 4.

Overall, based on considered literature, it was decided to test six "standard classifiers": Logistic Regression, Ridge, SVC, SGD, Decision Tree and Random Forest, and eleven NN-based classifiers: LSTM, Bi-LSTM, LSTM-Dropout, CNN-LSTM, Fast Text, RNN, RCNN, CNN, GRU, HAN, and BERT. Classifiers were tested using Count [19] and Tfidf [22] vectorizers. Ten NN-based classifiers (excluding BERT) were tested using Keras Embeddings [11] and, independently, Global Vectors for Word Representation (GloVe [16]) Embeddings. To combine results from multiple classifiers, a standard backpropagation neural network was used.

3 Data set and its preliminary analysis

Let us now describe the data set used in our work. As noted, it originates from the Kaggle competition: *Real or Not? NLP with Disaster Tweets. Predict which Tweets are about real disasters and which ones are not* [10]. The complete data set contains 10,876 tweets that were *hand classified* into: (1) *True*, i.e. notifying about actual disasters, and (2) *False*. This data set was (within Kaggle) divided into two files: *train.csv*, and *test.csv*. The *train.csv* contains 7613 rows, and 5 columns: *id*, *keyword*, *location*, *text*, *target*. The first 10 rows from the *train.csv* file are presented in Table 1.

Table 1. First rows from the *train.csv* file.

id	keyword	location	text	target
1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1
8	NaN	NaN	#RockyFire Update =¿ California Hwy. 20 closed...	1
10	NaN	NaN	#flood #disaster Heavy rain causes flash flood...	1
13	NaN	NaN	I'm on top of the hill and I can see a fire in...	1
14	NaN	NaN	There's an emergency evacuation happening now ...	1
15	NaN	NaN	I'm afraid that the tornado is coming to our a...	1

Here, the individual columns have the following meaning:

- *id* - does not contain any relevant information, so it will not be used;
- *keyword* - contains keywords, or *NaN* if the tweet does not include a keyword;
- *location* - contains location, or *NaN* if the tweet does not include location;
- *text* - contains text of the tweet,
- *target* - contains correct prediction for the tweet; *1* means that the tweet notifies about a disaster, and *0* if the tweet does not announce an emergency.

File *test.csv* contains *test data* that was to be used within the competition. This file contains 3263 rows and 4 columns: *id*, *keyword*, *location* and *text*, with the same meaning as above. Obviously, it does not include the *target* column.

Following the link to the Kaggle Leaderboard [9], it is easy to notice a number of submissions that have a perfect score. This is because of the data leakage, i.e. correct predictions for the "test tweets" appeared on the Internet. Hence, it can be reasonably claimed that some participants used this data to submit their "predictions". Moreover, it is **not possible** to establish, which results that are *not* 100% accurate, have been obtained by training actual classifiers. This is because it is easy to envision that some participants could have "produced" results "close to perfect", but not "stupidly perfect". Obviously, due to the leak, Kaggle cancelled this competition. Nevertheless, analyzing the Leaderboard it can be stipulated that the first "realistic result" is 0.86607 submitted by the "right_right_team" team. However, for obvious reasons, this claim is *only* a "reasonable speculation".

Overall, in what follows, all classifiers were trained using data from the *train.csv* file. Moreover, performance is reported in terms of the actual Kaggle score, calculated by applying the trained classifiers to the *test.csv* file. In this way, keeping in mind all, above mentioned, limitations of this approach, a baseline performance can be formulated.

3.1 Statistical analysis of the data set.

A Python program, inspired by [6], was developed to analyze key characteristics of the data set. We report these aspects of data that are known (from the literature; see, Section 2) to, potentially, influence the performance of the classifiers. To start on the most general level, in Table 2, class distributions for the *train data set* and the *test data set* are presented. It is easily seen that there are more "non disaster tweets" than "disaster tweets", in both data sets. However, their ratio (number of non disaster tweets divided by number of disaster tweets) is almost the same.

Table 2. Basic statistics of the *train* and *test* data sets

Distribution	*Train data set* (7613 - 100%)			*Test data set* (3263 - 100%)		
Type	Disaster	Non Disaster	Ratio	Disaster	Non Disaster	Ratio
Class	3243 (42,6%)	4370 (57,4%)	0,7421	1402 (43,0%)	1861 (57,0%)	0,7534
Keyword	3201 (42,0%)	4351 (57,1%)	0.7357	1386 (42,5%)	1851 (56,7%)	0,7488
Location	2177 (28,6%)	2903 (38,1%)	0,7499	936 (28,7%)	1222 (37,5%)	0,7660

Each data set, has a *keyword* column. In Table 2, it is shown how many tweets of each class have a keyword data (not all tweets were provided with keywords). It can be noticed that trends, for both data sets, match trends of class distribution. It thus can be concluded that there is no visible correlation between tweet keyword data and its class. Overall, data from the *keyword* column will be used in the training process.

Also, in Table 2, it is visible that a large number of tweets do not have the location data. However, the general trend is very similar to the class distribution (ratio of non disaster tweets to disaster tweets). Hence, it can be concluded that there is no immediate correlation between location data and tweets' class. This hypothesis was experimentally confirmed.

Data from Table 2 supports the conjecture that *train* and *test* data sets originated from the same data set (see, also [17]). This conjecture has been further experimentally verified and thus, in the remaining parts of this Section, we depict properties only for the *train data set*.

Let us start form the distribution of the number of characters in the disaster and non-disaster tweets, which is depicted in Figure 1.

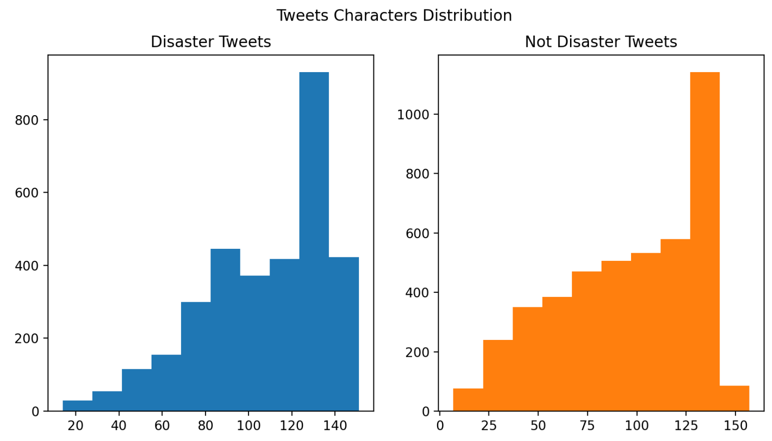

Fig. 1. Characters distribution for disaster and non-disaster tweets.

It can be seen that disaster tweets and the non-disaster tweets have quite similar character distributions. But, it is also fair to notice that, in general, non disaster tweets have more characters than the disaster tweets. This fact, however, has not been used explicitly in model training.

Next, let us compare distributions of numbers of words in disaster and non-disaster tweets (presented in Figure 2).

Here, it can be easily seen that the number of words distributions have similar shapes. Moreover, note that the scale for both graphs is the same. It should be kept in mind that the fact that non disaster tweets have slightly "taller bars" is caused by the fact that there are more tweets in this class.

Thus far we have not found substantial statistical differences between disaster and non disaster tweets. Situation is quite similar when the word length distribution is considered (see, Figure 3). While the scales of both figures are slightly different, the overall shape of the distribution is very much alike.

Let us now consider the top 10 stopwords found in disaster and non-disaster tweets. They have been captured in Figure 3.1.

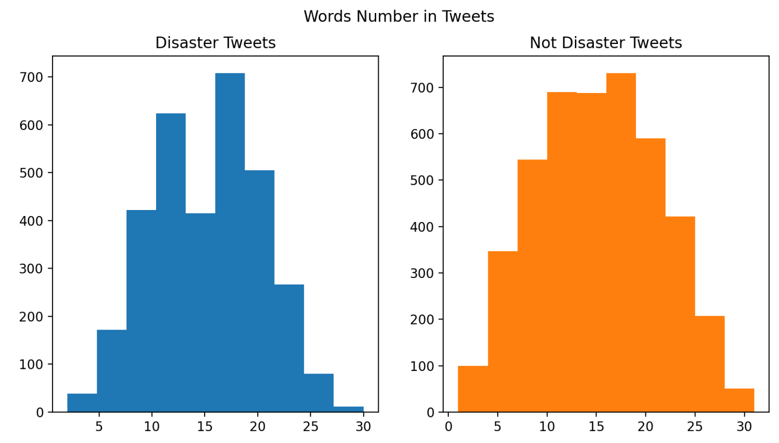

Fig. 2. Word number distribution for disaster and non-disaster tweets.

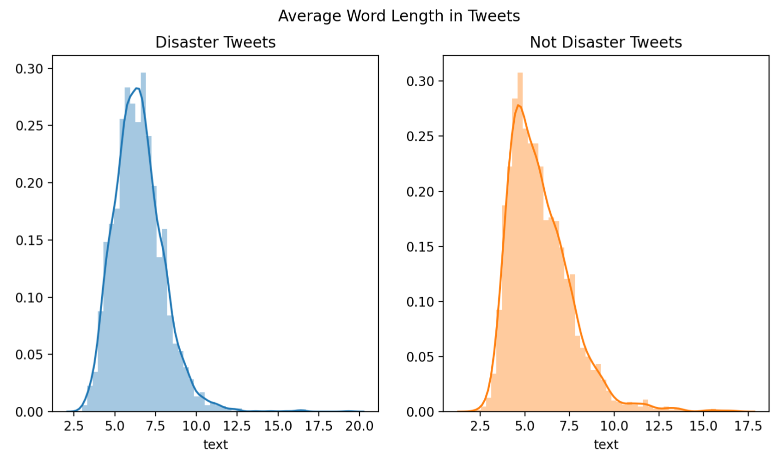

Fig. 3. Word length distribution for disaster and non-disaster tweets.

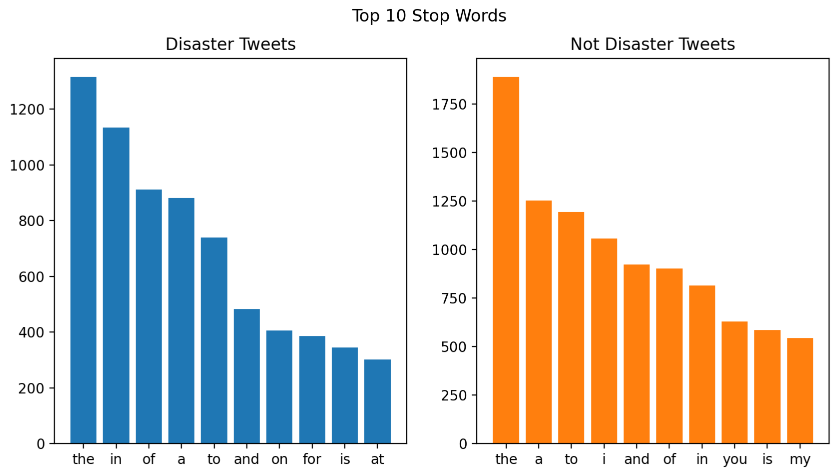

Fig. 4. Top 10 stopwords in disaster and non-disaster tweets.

In both cases, *the* stopword is the most popular one. The fact that it seems to be more popular in non disaster tweets is “normal”, because there are more tweets of this class. In the non disaster class, the next most popular stopwords are: *i*, *you*, *my*. Interestingly, these stopwords are not present in the disaster tweets class. There, the most popular stopwords (other than ‘*the*) are: *in*, *of*, *on*, *for*, *at*. These are descriptive stopwords that can be connected with the description of the details of the disaster. Therefore, stopwords may be useful in recognizing the nature of a tweet (and should not be removed).

As the last statistical comparison, let us consider punctuation symbols in both tweet classes (as illustrated in Figure 5).

It can be seen that both classes of tweets have the same order of top 3 punctuation signs (’-’, ’—’, ’:’). Hence, punctuation symbols do not differentiate between classes (and can be removed during the preprocessing phase of data preparation).

Finally, after analysing occurrences of the top 10 hashtags in the data set, it was noticed that there is a fairly small number of hashtags embedded in tweets, as compared to the total number of tweets. For example, in the *train data set* the top hashtag *#news* occurred less than 60 times, while the top not disaster tweet hashtag *#nowplaying* occurred around 20 times (in total). Thus, it can be conjectures that hastags will not bring much value to the classifier training. Therefore, it was decided to explicitly tag tweets that include a hashtag, but the hashtag itself will not be considered as a “different word” (e.g. *#news* and *news* will be treated as the same word).

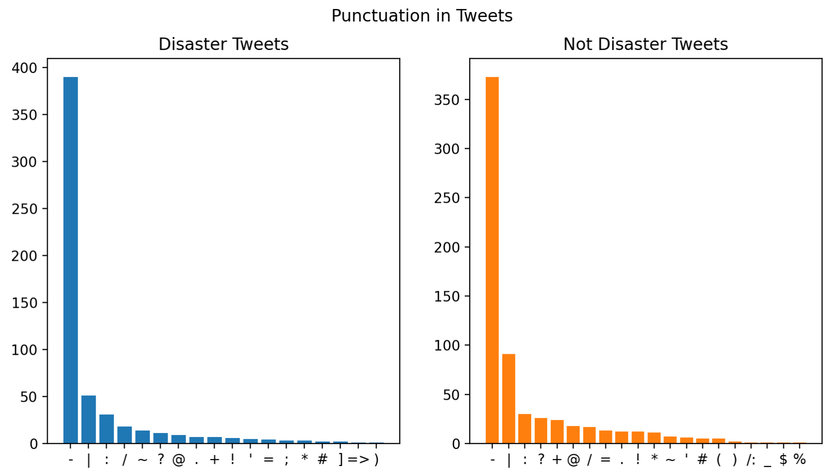

Fig. 5. Punctuation characters in distribution for disaster and non-disaster tweets.

4 Data preprocessing

Data preprocessing is a standard initial step of all data analytics/machine learning procedures. While the NLTK [15] is a popular natural language processing library, it had to be slightly augmented to deal with the tweets. Specifically, there are 17 configurable parameters in the *TweetPreprocessing.preprocess* method, created for the purpose of preprocessing. Therefore, 2^{17} different preprocessors can be created. Hence, it would be extremely time consuming to check the performance of all possible preprocessors, especially since the majority of them will perform poorly. This latter statement has been (partially) experimentally verified. Thus, it was decided to select, and "freeze" a set of operations performed by all of them, and thus substantially reduce the total number of possible preprocessors. Operations that have been selected to be used in all preprocessors were: (1) make all words lower letter, (2) remove slang words, (3) remove links, (4) remove Twitter users, (5) remove hashtags, (6) remove numbers. These operations were selected experimentally (it was found that they "always" improved the performance of classifiers). The last frozen parameter was (7) to not to perform stemming. This was an unexpected discovery, as stemming is one of key preprocessing techniques in NLP. After extensive experimentation with the remaining 10 parameters, 10 preprocessors have been selected. They are summarized in Table 3 (names of parameters are self-explanatory).

Note that six operations add flags to the string (e.g. *Hash Flag* and *Link Flag* set to "T" mean that *"My #string has a link https://mylink.com"* will be transformed into *"My #string has a link https://mylink.com $<$ hashtag $>$ $<$ link $>$"*). Here, it was found that the content of these items might not be

Table 3. Selected Preprocessing Algorithms

ID	*Link Flag*	*User Flag*	*Hash Flag*	*Number Flag*	*Keyword Flag*	*Location Flag*	*Remove Punct*	*Remove Stpwrds*	*Remove NotAlpha*	*Split Words*
PA01	T	T	T	T	F	F	T	T	T	T
PA02	F	T	T	F	F	F	T	T	T	T
PA03	T	T	T	T	T	T	T	T	T	T
PA04	T	T	T	F	F	F	T	T	T	T
PA05	F	F	F	F	F	F	T	T	T	T
PA06	T	T	T	T	F	F	F	F	F	T
PA07	T	T	T	T	F	F	F	F	F	F
PA08	F	F	F	F	F	F	F	F	F	F
PA09	F	F	F	F	F	F	T	F	F	F
PA10	T	T	T	T	F	F	T	F	F	F

important. For example, the link itself, usually has no useful information. However, the fact that the tweet contains a link, user, hashtag, numbers, or location, might be important. Note that the Global Vectors for word Representation (GloVe, [16]) has similar flags included. It has vector representations for *Link*, *User*, *Hash* and *Number* flags. While analysis reported in Section 3.1, suggested that *keyword* and *location* columns are not likely to be useful, these flags have been included for the testing purposes.

Preprocessors PA01, PA02, PA03, PA04 and PA05 were selected to test how the presence of *Link*, *User*, *Hash*, *Number*, *Keyword* and *Location* flags, influences performance of classifiers. Here, the remaining parameters are set to *T*.

Preprocessors PA06, PA07, PA10, and PA01, have similar configuration for the “flag presence parameters”, but different configurations of the remaining parameters. The four parameters *Link Flag*, *User Flag*, *Hash Flag* and *Number Flag*) are set to *T* to match the vector representations in GloVe. These preprocessors test how *Split Word*, *Remove Punctuation*, *Remove Stopwords*, and *Remove Not Alpha* parameters influence the performance of classifiers.

Finally, preprocessors PA08 and PA09, and PA05, have the “flag presence parameters” turned to *F*. They test how *Split Word*, *Remove Punctuation*, *Remove Stopwords*, and *Remove Not Alpha* parameters influence the performance of classifiers.

5 Experimental setup

Based on literature review, summarized in Section 2, 17 classifiers were selected: Logistic Regression, Ridge, SVC, SGD, Decision Tree and Random Forest, from the scikit-learn [20]; LSTM, Bi-LSTM, LSTM-Dropout, CNN-LSTM, Fast Text, RNN, RCNN, CNN, GRU, HAN and BERT utilizing Keras [12].

Apart from pairing classifiers with preprocessors, classifiers were tested for different parameter configurations, within each pair. Six Sklearn based classifiers were tested with the *Count* ([19]) and *Tfidf* ([22]) vectorizers, with different *ngrammar* ranges ([1,1], [1,2], [1,3]). The 10 NN-based classifiers (excluding BERT) were tested with Keras and GloVe Embeddings, with different hyper parameters. Experiments with BERT were conducted separately because it is a pre-trained classifier, requiring an embedding mechanism different from other NN-based classifiers. It should be noted that additional experiments have been performed to further explore possibility of building a "perfect classifier", but they were found to be much worse that these reported here.

We proceeded with exactly the same data split as in the Kaggle competition. The original data set (10,876 tweets) was split into standard 70-30% and 10-fold cross-validation was used. Specifically, for the cross-validation we split the *train data* into 90/10% (6447/716 tweets). Here, experiments were executed using StratifiedKFold [21] with 10 splits, and the seed equal to 7.

6 Experimental results

Let us now report and discuss obtained results. Note that the complete set of results, for all classifiers and preprocessors that have been experimented with, can be found at [18]. This is provided since the actual volume of generated data is such, that it cannot be properly depicted in the article itself.

6.1 Base classifiers and preprocessors

Let us start from considering combining preprocessing algorithms, with Ridge, Linear Regression, Logistic Regression, Random Forest, Decision Tree, SVC and SGD classifiers. They were tested with ten different preprocessing algorithms using *Tfidf* and *Count* vectorizers, with different grammar ranges ([1, 1], [1, 2], [1, 3]). Overall, 360 (6 classsffiers * 10 preprocessing algorithms * 2 Vectorizers * 3 ngrammars) tests cases have been completed, exploring all possible combinations of classifier and preprocessor pairs. In Table 4 best results obtained for each of the six classifiers are presented.

Table 4. Best results for classifiers not based on neural networks.

Classifier	Vectorizer [grammar range]	Algorithm ID	Score
Ridge	Tf-idf [1, 2]	PA08, PA09	0.80444
Logistic Regression	Count [1, 2]	PA04	0.80245
SVC	Count [1, 1]	PA03	0.80251
SGD	Tf-idf [1, 3]	PA08	0.80156
Random Forest	Tf-idf [1, 1]	PA02	0.78881
Decision Tree	Count [1, 2]	PA05	0.76332

As can be seen, the Ridge classifier performed the best. Moreover the same result was obtained in combination with two different preprocessors (PA8 and

PA9). Reaching beyond results presented in Table 4, when considering all experimental results, no specific preprocessing algorithm has been found to be a "definite champion". Hence, a "relative strength" of preprocessors was calculated. For each preprocessor, for each classifier, the best three results were considered. If a given preprocessor was the best for a given classifier, it received 3 points, the second place scored 2 points, while the third place 1 point (see, Table 5).

Table 5. Preprocessing algorithms relative score

Algorithm	Positions Points	Score
PA09	3 + 1 + 1 + 3	8
PA04	3 + 2 + 2 + 1	8
PA02	1 + 2 + 3 + 2	8
PA08	3 + 3	6
PA03	1 + 3	4
PA01	2 + 1	3

Using the proposed scoring method (while keeping in mind its limitations), the best scores were achieved for **PA09**, **PA04** and **PA02** preprocessors. However, it is the **PA09** that can be considered a winner, as it earned two "first places". The second best was **PA04** (one first place, when combined with the Logistic Regression classifier). This preprocessor adds link, user, hash flags and removes punctuation and stopwords. The third best was **PA02**, which is very similar to **PA04**, except the **PA04** adds link flag but **PA02** does not.

To complete the picture, let us now present the 15 best preprocessor+classifier pairs (in Table 6).

From results presented thus far it can be concluded that the three "leaders among non-NN-based classifiers" were: Ridge, SVC, and Logistic Regression. They performed best, but their best results were obtained when they were paired with different preprocessing algorithms. Henceforth, there is no preprocessor that could be said that it performs best with every classifier. This may indicate that, in general, for different problems, different classifiers require different preprocessors to reach full potential.

6.2 Preprocessors and Neural Network based classifiers

Let us now report on the results of the tests that combined preprocessors and the ten different NN-based classifiers: LSTM, LSTM-Dropout, Bi-LSTM, LSTM-CNN, Fast Text, RCNN, CNN, RNN, GRU and HAN. All these classifiers were tested with Keras and GloVe embeddings, with different hyper-parameters. Note that results of BERT classifier are reported separately, because BERT requires an embedding mechanism different from other NN-based classifiers.

Table 6. Top 15 preprocessor+classifier pairs

Classifier	Algorithm ID	Score
Ridge	PA08	0.80444
Ridge	PA09	0.80444
SVC	PA03	0.80251
Logistic Regression	PA04	0.80245
SVC	PA04	0.80233
Ridge	PA02	0.80211
Logistic Regression	PA04	0.80211
Ridge	PA09	0.80196
Ridge	PA08	0.80196
SVC	PA10	0.80165
SVC	PA07	0.80162
SGD	PA08	0.80156
Ridge	PA02	0.80153
Logistic Regression	PA02	0.80153
SVC	PA06	0.80147

Let us start with the NN-based classifiers utilizing Keras embeddings. In Table 7 presented are top pairs scores of classifier + preprocessor.

Table 7. Top scores for the neural network based classifiers and Keras embeddings

Model	Preprocessor	Score
LSTM	PA09	0.79684
LSTM_DROPOUT	PA01	0.79782
BI_LSTM	PA04	0.79746
LSTM_CNN	PA04	0.79384
FASTTEXT	**PA07**	**0.80374**
RCNN	PA10	0.79418
CNN	PA07	0.79994
RNN	PA01	0.79859
GRU	PA06	0.80055
HAN	PA06	0.80172

It can be easily noticed that the average scores in Table 7 are quite similar to each other. Moreover, as in the case of non-NN approaches, different classifiers performed best when paired with different preprocessors. As a matter of fact, not a single preprocessor occurs in this table more than two times. Again, there is no pair of preprocessor and NN-based classifier, which stands out significantly among others. However, the best average score was achieved with the PA07 preprocessor and the Fast Text NN-based classifier.

Let us now consider NN-based classifiers utilizing GloVe [16] embeddings. In Table 8 top pairs of classifiers and preprocessors are presented.

Table 8. Top average scores for the neural network based classifiers and GloVe embeddings

Model	Preprocessor	Score
LSTM	PA04	0.80996
LSTM_DROPOUT	PA02	0.81257
BI_LSTM	PA04	0.81039
LSTM_CNN	PA04	0.80610
FASTTEXT	PA04	0.81229
RCNN	PA09	0.80478
CNN	PA10	0.80622
RNN	PA02	0.80788
GRU	**PA04**	**0.81444**
HAN	PA06	0.80975

In general, it can be seen that average scores, in Table 8, are higher than these in Table 7. This suggests that using GloVe embeddings increased the performance of considered classifiers. Also, it can be seen that the PA04 preprocessor was at the top for 5 out of 10 models. Moreover, the GRU classifier, paired with the PA04 preprocessor, had the top average score. It is safe to state that the PA04 preprocessor stands out as the best pair for classifiers, which use the GloVe embeddings. One of possible reason, why this preprocessor was "the best", might be that it adds three flags to the tweets (user, link, and hashtag, which also have representation in GloVe). Moreover it removes "noise", such as stopwords, punctuation, and non-alphabetic words. In this way it it may be the best match with the GloVe embeddings used in this series of experiments. However, this is just a stipulation, which has not been further verified and as such, needs to be taken with a grain of salt.

6.3 Preprocessors and BERT classifier

In this section, experiments with the BERT-Large model, from the TensorFlow Models repository, on GitHub [8], are discussed. Here, BERT uses $L = 24$ hidden layers of size of $H = 1024$, and $A = 16$ attention heads. BERT was tested with the ten preprocessors, and without the preprocessing phase. The average scores are presented in Table 9.

It can be easily noticed that the average scores obtained by BERT classifier are much higher than the average scores obtained with all other tested classifiers. The best average score is achieved when BERT was combined with the PA08 preprocessor. It should be noted that the best individual score, achieved for any experiment, was 84,064% accuracy. This result would have landed at the 79th

Table 9. BERT averages scores for different preprocessors

Model	Preprocessor	Score
BERT	None	0.82991
BERT	PA01	0.82109
BERT	PA02	0.82216
BERT	PA03	0.82182
BERT	PA04	0.82155
BERT	PA05	0.81649
BERT	PA06	0.83154
BERT	PA07	0.83252
BERT	**PA08**	**0.83469**
BERT	PA09	0.82832
BERT	PA10	0.82581

place, out of 3402 submissions, in the Kaggle Leaderboard (if scores, which are better than 98%, were excluded, as potentially fraudulent). While, the average score is much better in capturing the overall strength of the approach, it is also obvious that the scores reported for the competition were the best ones actually achieved in any single run. Therefore, it can be stated that the 79th place, claimed above, is somewhat realistic. This is also the best score we managed to obtain my using any single classifier (with its best sidekick preprocessor).

7 Combining predictions form multiple classifiers

Let us now consider the possibility of "combining suggestions" from multiple classifiers (paired with their best preprocessors). The goal, here, is to deliver results that would be more accurate than these from any single classifier+preprocessor pair. Specifically, the top six classifiers were taken from those listed in Table 4. Moreover, top ten NN-based classifiers, based on the Keras embeddings, were taken from those listed in Table 7, as well as top ten NN-based classifiers, based on GloVe embeddings, from Table 8. Finally, the best BERT pair from Table 9 was selected. As a result, 27 pairs of classifier+preprocessor were selected. For clarity, they are listed in Table 10.

Table 10: Top 27 Pairs of Classifier and Preprocessor

Model/Classifier	Preprocessor
BERT	PA08
LSTM-GloVe	PA04
LSTM_DROPOUT-GloVe	PA02
BI_LSTM-GloVe	PA04
LSTM_CNN-GloVe	PA04
FASTTEXT-GloVe	PA04

Continuation of Table 10

Classifier	Preprocessor
RCNN-GloVe	PA09
CNN-GloVe	PA10
RNN-GloVe	PA02
GRU-GloVe	PA04
HAN-GloVe	PA06
LSTM	PA09
LSTM_DROPOUT	PA01
BI_LSTM	PA04
LSTM_CNN	PA04
FASTTEXT	PA07
RCNN	PA10
CNN	PA07
RNN	PA01
GRU	PA06
HAN	PA06
RIDGE	PA08
SVC	PA03
LOGISTIC_REGRESSION	PA04
SGD	PA08
DECISION_TREE	PA05
RANDOM_FOREST	PA02

Each classifier+preprocessor pair, from Table 10, was trained using 70% of the (original Kaggle) *train data set*, and validated against the remaining 30%. Note that, for each of the 27 individual classifiers, the same data split (exactly the same data used in both the training and the validation sets) was used, during the training phase.

Each of the trained classifiers produced its own predictions, for the tweets in the validations set, resulting in 27 independent lists of "suggestions". These 27 suggestion vectors were transformed into input for the neural network: $[v_1, v_2..., v_n]$, where v_i − is the list of 27 predictions per tweet; $i = 1 \ldots n$; n − is the number of tweets in the validation set (30% of the train data set). Here, a standard backpropagation neural network, with 27 input neurons, 14 neurons in the hidden layer and a single output neuron, was used. After training the neural network on this input (with the same split 70%/30%), the average results did only barely beat the score of the best standalone BERT+ preprocessor pair. Specifically, BERT classifier (combined with PA08 preprocessor), had the best recorded accuracy of 0.83604. This has to be compared with the best single result of the meta-classifier, which was better by 0.004. This improvement would not have boost the score in the Kaggle competition. The average score of the best BERT+preprocessor pair was 0,83469, while the improvement (of the average score) of the meta-classifier was 0.00071. Overall, it can be concluded that combining predictions from multiple classifiers has potential, but requires further investigation.

8 Concluding remarks

Let us now summarize the main conclusions for the research discussed within the scope of this paper.

- Based on literature analysis, and comprehensive experimentation, ten preprocessors have been selected. All of them had (the same) 7 out of 17 possible tweet preprocessing parameters/operations "frozen". Frozen parameters have "always" improved the performance of classifiers.
  Preprocessors PA01, PA02, PA03, PA04 and PA05, tested how the presence of *Link*, *User*, *Hash*, *Number*, *Keyword* and *Location* flags, influences performance of classifiers. The remaining parameters were set to T.
  Preprocessors PA06, PA07, PA10, and PA01, have similar configuration for the "flag presence parameters", but different configurations of the remaining parameters. Parameters *Link Flag*, *User Flag*, *Hash Flag* and *Number Flag*) were set to T. among others, to match the vector representations in GloVe. These preprocessors test how *Split Word*, *Remove Punctuation*, *Remove Stopwords*, and *Remove Not Alpha* parameters influence the performance of classifiers.
  Finally, preprocessors PA08 and PA09, and PA05, have the "flag presence parameters" turned to F. They test how *Split Word*, *Remove Punctuation*, *Remove Stopwords*, and *Remove Not Alpha* parameters influence the performance of classifiers in this context.
- Six "base" classifiers, using different vectorizers and ngrammars, combined with selected preprocessors, were tested. It was discovered that the best average of 10-fold cross validation accuracy of 0.80444 was achieved by the Ridge classifier, utilizing Tfidf vectorizer with [1, 2] ngrammar range, independently with two preprocessors: PA08 and PA09 (table 6). These two preprocessors are almost the same, except that PA08 does not remove punctuation from tweets, while PA09 does. Overall, the top three classifiers were Ridge, SVC and Logistic Regression. Moreover, the worst results were achieved by the Decission Tree and Random Forest classifiers (regardless of the applied preprocessing). Top three preprocessors were PA09, PA04 and PA02, as they obtained the same relative score (Table 5). However, it cannot be said that one of them is the best one overall, and that every classifier performs the best with it.
- Ten NN-based classifiers, using Keras embedddings, were combined with selected preprocessors and tested. It was found out that Fast Text NN-based classifier, combined with PA07 preprocessor, had the best average score of 0.80374 (Table 7). Average scores of all classifiers were "close to each other". However, they were achieved when combined with different preprocessors. Hence, again, no preprocessor was found to perform best with all NN-based classifiers.

- The same ten NN-based classifiers, combined with sepected preprocessors and using GloVe embeddings were tested. All NN-based classifiers with GloVe embeddings produced better scores than same classifiers with Keras embeddings. The top average score of 0.81444 was achieved by GRU NN-based classifier combined with PA04 preprocessor (Table 8). Again, all classifiers obtained similar results. However, in this case, the PA04 preprocessor produced 5 top averages results out of 10 NN-based classifiers, and was a clear "winner".
- BERT classifier had the best average score among all classifiers. The best average score 0.83469 was achieved with the PA08 preprocessor (Table 9). The best single score with BERT classifier was achieved with the same preprocessor, and was 0.84064. This score would result in reaching 79th place, out of 3402 submissions, in the Kaggle Leaderboard (excluding scores better than 98% that can be assumed to be compromised).
- Finally, the 27 pairs of best classifiers + preprocessors (from Table 10) were combined, using a simple neural network, into a meta-classifier. This approach did overcome best single BERT model accuracy by maximum of 0.004.

The single most interesting research direction, following from the above-reported results, could be to further investigate possibility of combining predictions of multiple classifiers, to achieve better overall accuracy.

References

1. Ajao, O., Bhowmik, D., Zargari, S.: Fake news identification on twitter with hybrid cnn and rnn models. in proceedings of the international conference on social media & society. SMSociety (2018), https://arxiv.org/ftp/arxiv/papers/1806/1806.11316.pdf, dOI: 10.1145/3217804.3217917
2. Ashktorab, Z., Brown, C., Nandi, M., Mellon, C.: Tweedr: Mining twitter to inform disaster response. Proceedings of the 11th International ISCRAM Conference – University Park (May 2014), http://amulyayadav.com/spring19/pdf/asht.pdf
3. CrisisLex: Crisislex data set. https://github.com/sajao/CrisisLex/tree/master/data/CrisisLexT26
4. CrisisNLP: Crisisnlp data set. https://crisisnlp.qcri.org/
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (May 2019), https://arxiv.org/pdf/1810.04805.pdf, arXiv: 1810.04805
6. ES, S.: Basic eda, cleaning and glove. https://www.kaggle.com/shahules/basic-eda-cleaning-and-glove
7. Ganzha, M., Paprzycki, M., Stadnik, J.: Combining information from multiple search engines - preliminary comparison. Information Sciences **180**(10), 1908–1923 (2010), dOI: 10.1016/j.ins.2010.01.010
8. Google: Bert github. https://github.com/google-research/bert
9. Kaggle: Kaggle competition leaderboard. https://www.kaggle.com/c/nlp-getting-started/leaderboard
10. Kaggle: Kaggle competition: Real or not? nlp with disaster tweets. predict which tweets are about real disasters and which ones are not. https://www.kaggle.com/c/nlp-getting-started/overview

11. Keras: Keras embedding layer. https://keras.io/api/layers/core_layers/embedding/
12. Keras: Keras library. https://keras.io/
13. Kursuncu, U., Gaur, M., Usha Lokala, K.T., Sheth, A., Budak Arpinar, I.: Predictive analysis on twitter: Techniques and applications. Kno.e.sis Center, Wright State University (Jun 2018), https://arxiv.org/pdf/1806.02377.pdf, arXiv: 1806.02377
14. Ma, G.: Tweets classification with bert in the field of disaster management. Department of Civil Engineering, Stanford University (2019), https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15785631.pdf
15. NLTK: Nltk library. https://www.nltk.org/
16. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/
17. Plakhtiy, M.: Applying machine learning to disaster detection in twitter tweets. https://drive.google.com/file/d/1k2BGDn3t76rQjQIMA2GaRIzSFLXwnEIf/view?usp=sharing
18. Plakhtiy, M.: Results on google drive. https://docs.google.com/spreadsheets/d/1ePODdEMxzNLT6ecfdN5Kf5ctK1BSYNgoq1YHylSVDLc/edit?usp=sharing
19. SkLearn: Count vectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
20. SkLearn: Scikit-learn library user guide. https://scikit-learn.org/stable/user_guide.html
21. SkLearn: Stratifiedkfold. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
22. SkLearn: Tfidf vectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
23. Stadnik, J., Ganzha, M., Paprzycki, M.: Are many heads better than one – on combining information from multiple internet sources. Intelligent Distributed Computing, Systems and Applications pp. 177–186 (2008), http://www.ibspan.waw.pl/~paprzyck/mp/cvr/research/agent_papers/IDC_consensus_2008.pdf, dOI: 10.1007/978-3-540-85257-5_18
24. Thanos, K.G., Polydouri, A., Danelakis, A., Kyriazanos, D., Thomopoulos, S.C.: Combined deep learning and traditional nlp approaches for fire burst detection based on twitter posts. IntechOpen (April 2019), https://doi.org/10.5772/intechopen.85075, dOI: 10.5772/intechopen.85075
25. Wikipedia: Logistic regression. https://en.wikipedia.org/wiki/Logistic_regression
26. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., Hovy, E.: Hierarchical attention networks for document classification. Carnegie Mellon University, Microsoft Research, Redmond (Jun 2016), https://www.cs.cmu.edu/~./hovy/papers/16HLT-hierarchical-attention-networks.pdf
27. Zhang, C., Rajendran, A., Abdul-Mageed, M.: Hyperpartisan news detection with attention-based bi-lstms. Natural Language Processing Lab, The University of British Columbia (2019), https://www.aclweb.org/anthology/S19-2188.pdf
28. Zubiaga, A., Hoi, G.W.S., Liakata, M., Procter, R.: Pheme data set. https://figshare.com/articles/PHEME_dataset_of_rumours_and_non-rumours/4010619