1. Methodology

Let us suppose that a dataset of n documents D_i is given. Each document consist of twitter message text and corresponding label y_i. Labels are binary values, where one identifies crisis-related message and zero - non-related ones. Each text represents raw string, therefore preprocessing is done to remove special characters and leave just blank words. Also, each text should be converted into a vector of features $x_i \in \mathbb{R}^m$. We will evaluate a set of well-known algorithms, such as different types of word-level embeddings, to perform following conversion. The embedding algorithms should represent the meaning of each text as a vector in m-dimensional vector space. Then, the dataset will be splitted into two unequal parts, the biggest is for training proposes, the other - for testing. Assuming that a set of machine-learning based classification algorithms is given, the objective is to train the model $f(x_i)$ and then perform evaluation on the test part into to receive outputs $\hat{y}_i$ and evaluate the quality of selected classification algorithm by calculating quality function $Q(Y, \hat{Y})$ and repeate the procedure for each classification and embedding algorithm from our set to choose a pair of best ones.

1.1. Embeddings

The objective of text embedding is to obtain text's meaning and represent them as a m-dimensional vector. There are two major branches of embedding algorithms: word-level [2] and sentence-level[1]. Main idea is to train neural network to estimate difference and similarity between words/sentences. Example: let $V(t_i)$ be the embedding algorithm:

$$distance[V("the\_cat\_meows"), V("the\_cat\_sleeps")] <$$

$$distance[V("the\_cat\_meows"), V("stocks\_are\_overheated")]$$

We will feed the preprocessed texts into embedding algorithms. For the case of word-level embeddings we will take the average vector of all the words :

$$V("the\_cat\_meows") = \frac{1}{3}[V("the") + V("cat") + V("meows")]$$

For the sentence-level embeddings the entire document will be represented as one sentence.

1.2. Quality measurment

As far as we are supposed to choose the best embedding and classification algorithms for our domain, quality of each pair should be measured to compare them. Let X_{train}, Y_{train} are the training documents x_i and corresponding labels $y_i, i = \overline{1, n_{train}}$. Also, let X_{test}, Y_{test} bare the test part of documents x_j and labels $y_j, j = \overline{1, n_{test}}$. After training embeddings and classification algorithm, the evaluation will be performed on a X_{test} to get the predictions:

$$Y_{pred} = \{f(x_i) | \forall x_i \in X_{test}\}$$

Then, the value of quality function $Q(Y_{pred}, Y_{test})$ will be computed. To avoid receiving optimitically overqualified results, we will perform k-fold cross-validation. We will present the mean and standard deviation of measurments from each split as a quality of given algorithm.

2. Experimental set-up

2.1. Dataset

We choose the CrisisLexT6 dataset for embedding and classification evaluation. That dataset consist of 60000 tweets from 6 major crisis situations, labeled by relatedness. The dataset comes splitted into 6 files, each for corresponding crisis, therefore the concatenation is done. Labels are represented as 'off-topic' for non related tweets and 'on-topic' for related and several tweets comes unlabeled, thus we have removed them from the dataset and converted labels to 1 - 0 respectively. We select K for k-fold cross-validation to be 5, so each block contains 12000 documents.

2.2. Preprocessing

Regular expressions is used to clean-up the data, remove the special characters and represent hashtags, mentions, emojis and urls as single tokens. Firstly, tweet's text is tokenized by NLTK's[3] TweetTokenizer. Then, we employ python's component "re" to evaluate regular expressions on tokens.

2.3. Classification algorithms

Statistically based and deep-learning algorithms seems compatitve at our domain. Here is a list of algorithms we choose to compare:

- Logistic Regression, proposed by Liblinear developers [5] with interface in Scikit-learn.
- Random Forest, proposed in [4], implemented at Scikit-learn.
- Gradient Boosted Decision Trees, implemented at LightGBM [6].
- Fully-connected neural nework

- CNN for text classification, which is briefly described in [8].
- C-LSTM, proposed by [15].

2.3.1. Neural networks

The **fully-connected network** is a simple 2-layer perceptron with dropout in the middle. The first layer activation is Rectified Linear Unit [7], when the outputs of last layer is passed through Softmax. The network is implemented using the Pytorch[9] deep learning framework.
The **Convolutional Neural Network for sentence classification**, proposed by Yoon Kim, is employed to classify documents, where each word represents it's own embedding vector, instead of taking single average vector for the whole text.Batches comes as an array of vocabulary words indices, padded to the length of 43 words per document, therefore the first layer is embedding, which matrix is initialized according to evaluated embedding algorithm. The main idea is to extract high-level contextual fetures by a set of independent convolutional layers with different kernel sizes. Then, extracted feature vectors comes to max pooling layer to reduce dimensionality. Next, vectors are stacked into single one and feeded to fully-connected layer to make final prediction of document's class. Activation functions for convolutional and fully-connected layer are set to ReLU and Softmax respectively.
C-LSTM network's main goal is to exctract close-context features by convolutional layer, and then feed them to LSTM layer to find out time-relevant dependencies. In general, C-LSTM architecture consist of Embedding layer, followed by 1-d convolution with relu activation and max pooling after and lstm layer. The final predictions are made by two linear layers with hyperbolic tangent and softmax activation.

2.3.2. Hyperparameters

- **Logistic Regression** We set-up logistic regression with l2 penalty, stopping criteria tolerance equal to 0.0001 and inverse regularization strength to 1.0. Bias is added as constant feature.
- **Random Forest** is employed with 1000 estimators and no limits to maximum number of features and tree depth. Gini criterio is setted up to measure the quality of split. Minimum number of samples per split and per leaf are 2 and 1 respectively.
- **Gradient Boosted Decision Trees** with maximum tree depth of 20, 11 as number of leaves, learning rate of 0.05. feature fraction 0.9, bagging fraction 0.8 and frequency of 5. Number of estimators were set to 4000 with early stopping for 200.

- **Fully-Connected Network** is setted up with hidden layer size 256 and dropout with probability 0.5. Training were performed for 10 epochs with binary cross entropy as loss criterion. Optimization is done by Adam algorithm with learning rate 0.0001 and 0 weight decay. Batch size were set to 256.

- For **CNN for sentence classification** we use filter windows of [3, 4, 5] with 512 feature maps each. Dropout prob of 0.5. Optimization is done by Adam with learning rate 0.0001 and binary cross entripy as loss. Batch size is set to 128 and vocabulary size were 10001. Training is performed for 10 epochs with early stopping for 3 epochs.

- **C-LSTM** is configured with filter window of 3 with 128 feature maps, max pooling kernel of 2, LSTM hidden size of 80 and dropout with prob 0.1. First linear layer hidden size is 60. Training was performed for 15 epochs with early stopping on 1 epoch. Adam with 0.0001 learning rate is employed as optimizer and binary cross entropy as loss function. We use batch size of 64 and vocabulary size same to CNN for text classification.

2.4. Embeddings

There is a set of embedding algorithms we choosed to compare. The Fasttext algorithm, proposed in [10], is used in two variants: trained on our own dataset, and pre-trained on english wikipedia corpus. Another embeddings we choose to compare were GloVe, proposed in [11], pretrained on CommonCrawl corpora with output dimensionality 300 and on large corpus of tweets, with dimensionality 200. Also, we choose to train Word2Vec embedding algorithm, described in [2], on our dataset.
To compare word-level embeddings with sentence-level embeddings, we select Infersent[1] embeddings, which are comes pretrained on natural language inference data. The dimensionality of ouput vector is 4096. Each document is threated as single sentence, thus, we have no way to test thet embeddings on CNN and C-LSTM. Also, due to RAM limitations, we have not tested InferSent embeddings with Gradient Boosted Decision Trees.

2.5. Quality measurment and hardware set-up

As far as dataset we train on is well-balanced for classes, the accuracy and F1 score is choosed to be our quality functions. We found reasonable to present the measuments for dummy baseline(all the test samples are predicted as class 1) to ensure, that compared algorithms makes significantly better predictions, than constant.
Experimental machine set-up is Nvidia Tesla K20 GPU with 6 GB of memory, two Intel Xeon CPUs, with 16 cores each and 64 GB of RAM.

3. Results and Discussion

Results of our comparasion are listed on table 1 - 3. We discovered several insights into problems with processing and analyzing crisis and twitter specific lexicon. Regarding to results, we are able to state on following opinions:

- In most cases, it is better to train embeddings on common text corpora instead of twitter specific one. Indead, GloVe embeddings, pretrained on a Common Crawl corpora shows better results, than any other word-level embeddings. Also, sentence-level embeddings, pretrained on non-specific natural language inference data, shows better results at all. In general, it seems reasonable that crisis-related lexicon differs from common twitter lexicon, and tends to be closer to common lexicon. On the other hand, there is a lack of publicily available twitter data to pretrain embeddings. The twitter corpora, that were given to pretrain GloVe embeddings we use, is only 4 billion words long, which is uncomparable to CommonCrawl corpora size of 840 billion words.
- Sentence level embeddings are better than average word vectors. In our opinion, taking an average of all the document's words embedding vectors can probably blur the real meaning of text. But the InferSent embeddings, which are based on NLI data and BiLSTM encoders, treats sentence as signle entity and performs more general projection process. But the higher dimensionality (required to make accurate projections) makes it harder to use several classification algorithms.
- Neural networks prediction quality is stable. As shown by "* Std" columns in our table, all the neural networks have lower standard deviation of accuracy and f1 than statistically based algorithms(except GBDT). That means that we will receive stably good results all the time, with no dependency to context and training data.

All the previous researches [12, 13, 14] threats the CrisisLexT6 as 6 different datasets, each for corresponding crisis. While in some crisises Naive Bayes classifier with domain adoptation algorithm, presented in [14], shows better accuracy than the mean accuracy of our best pair (CNN for text classification + Fasttext, thained on our dataset), their mean accuracy among all the observations are 0.02 lower than ours.

References

[1] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, A. Bordes, Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. arXiv preprint arXiv:1705.02364.

Table 1. Accuracy

	FstMean	FstStd	FsWikiMean	FsWikiStd	GlCCMean	GlCCStd	GlTwtMean	GlTwtStd	W2VMean	W2VStd	InfStMean	InfStStd	Baseline
LogReg	0.8772	0.0690	0.8270	0.0757	0.8846	0.0476	0.8486	0.0613	0.8897	0.0573	0.8930	0.0439	0.5434
Random Forest	0.8733	0.0747	0.8298	0.0853	0.8762	0.0627	0.8452	0.0825	0.8776	0.0720	0.8975	0.0425	0.5434
GBDT	0.9113	0.0009	0.8912	0.0020	0.9256	0.0018	0.8912	0.0020	0.9145	0.0027	N/A	N/A	0.5434
FullyConnected	0.9027	0.0036	0.8599	0.0041	0.9162	0.0039	0.8718	0.0029	0.9058	0.0020	0.9092	0.0026	0.5434
CNN	**0.9392**	0.0033	0.9296	0.0034	0.9346	0.0027	0.9230	0.0014	0.9248	0.0019	N/A	N/A	0.5434
CLSTM	0.9153	0.0025	0.9159	0.0052	0.9170	0.0050	0.9088	0.0072	0.9191	0.0051	N/A	N/A	0.5434

Table 2. F1

	Fst Mean	Fst Std	FstWiki Mean	FstWiki Std	GlCC Mean	GlCC Std	GlTwt Mean	GlTwt Std	W2V Mean	W2V Std	InfSt Mean	InfSt Std	Baseline
LogReg	0.8743	0.0848	0.8256	0.0921	0.8861	0.0528	0.8514	0.0693	0.8899	0.0671	0.8945	0.0493	0.7041
Random Forest	0.8693	0.0952	0.8233	0.1119	0.8744	0.0741	0.8399	0.1052	0.8744	0.0887	0.8945	0.0493	0.7041
GBDT	0.9171	0.0009	0.8984	0.0014	0.9306	0.0013	0.8986	0.0023	0.9201	0.0026	N/A	N/A	0.7041
FullyConnected	0.9099	0.0037	0.8684	0.0039	0.9223	0.0037	0.8808	0.0028	0.9120	0.0021	0.9088	0.0021	0.7041
CNN	**0.9432**	0.0034	0.9343	0.0035	0.9389	0.0029	0.9272	0.0017	0.9296	0.0017	N/A	N/A	0.7041
CLSTM	0.9211	0.0022	0.9222	0.0036	0.9226	0.0061	0.9153	0.0049	0.9230	0.0055	N/A	N/A	0.7041

[2] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space

[3] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[4] L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.

[5] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9(2008), 1871-1874.

[6] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In Advances in Neural Information Processing Systems (NIPS), pp. 3149-3157. 2017.

[7] Vinod Nair and Geoffrey Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. ICML. 2010

[8] Yoon Kim. Convolutional Neural Networks for Sentence Classification. http://arxiv.org/abs/1408.5882

[9] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. Automatic differentiation in PyTorch. NIPS-W. 2017

[10] P. Bojanowski*, E. Grave*, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics. 2017. Vol. 5. pp 135-146.

[11] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. 2015.

[12] Roy Chowdhury S, Purohit H, Imran M. D-sieve: a novel data processing engine for efficient handling of crises-related social messages. InProceedings of the 24th International Conference on World Wide Web 2015 May 18 (pp. 1227-1232). ACM.

[13] Zhang S, Vucetic S. Semi-supervised discovery of informative tweets during the emerging disasters. arXiv preprint arXiv:1610.03750. 2016 Oct 12.

[14] Li H, Caragea D, Caragea C, Herndon N. Disaster response aided by tweet classification with a domain adaptation approach. Journal of Contingencies and Crisis Management. 2018 Mar;26(1):16-27.

[15] Zhou C, Sun C, Liu Z, Lau F. A C-LSTM neural network for text classification. arXiv preprint arXiv:1511.08630. 2015 Nov 27.