Machine Learning A 2021/2022 Final Project
Binary Classification: Disaster Tweets

Elisa Tremolada
elisa.tremolada@studenti.unipd.it

February 4, 2022

1 Introduction

This project builds on the Kaggle challenge *"Natural Language Processing with Disaster Tweets"*. Its main objective is to perform binary classification using machine learning algorithms on a natural language text dataset.

The task at hand is that of classifying tweets which concern a real disaster (e.g. flooding, suicide bombing etc.) and tweets which don't, but still use language about disasters in order to emphasize their message (e.g. "This party is on fire"). It may seem like a trivial exercise, but nowadays many agencies (e.g. disaster relief organizations and news agencies) use Twitter in order to get real-time information about disasters. Therefore, a classification algorithm discriminating between tweets which are relative to an emergency and tweets which aren't could be a useful practical tool to help them promptly intervene where needed.

The goal of this project is then twofold: firstly, Natural Language Processing techniques are applied in order to convert natural language text data into a machine-readable form; secondly, various machine learning methods are employed in order to discriminate between *real* and *fake* "disaster tweets" as accurately as possible. The experiments conducted show that best classification performance is obtained with a **logistic regression** algorithm and a tf-idf base vectorization of the data.

2 Dataset and Preprocessing

The dataset provided is composed of 10,000 tweets that were hand classified between "disaster" ones (class 0) and "non-disaster" ones (class 1). It is divided between a *training set*, made up of 7163 tweets, and a *test set*, made up of 3263 tweets.

Since the data comes from a Kaggle challenge, test set class labels have been removed. Thus, exploratory data analysis is conducted only on the training set, while preprocessing is applied to both. The original dataset has **5 columns**: **id** (a unique identifier for each tweet), **location** (the location the tweet was sent from), **keyword** (a particular keyword from the tweet), **text** (the text of the tweet) and **target** (denotes whether a tweet is about a real disaster or not, using 1 and 0 labels, respectively).

Classes in the training set are slightly imbalanced, but not so much as to be talking about an imbalanced classification problem: there are 3271 *real* disaster tweets 4342 *fake* ones.

2.1 Feature selection

Firstly, the column **id** was dropped, since it gave us no useful information for classification. Secondly, exploration of the **location** column showed that 30% of values for this column are missing, while the remaining ones are reported in many different string formats (e.g. city name, country name, both...), making it very difficult to convert these text values into a machine-readable format. Thus, the column was dropped for simplification. The **keyword** column was kept as a feature, since both training and test set share the same keywords and, for the training set, the top-10 most frequent keywords for real disaster tweets differ from the top-10 most frequent keywords for fake ones, highlighting the usefulness of this feature for classification (see Figure 1).

disaster tweets	non-disaster tweets
outbreak	bodybags
derailment	armageddon
wreckage	harm
oil spill	deluge
typhoon	ruin
debris	wrecked
rescuers	explode
suicide bomb	siren
suicide bombing	twister
evacuated	fear

Figure 1: Top 10 keywords for both disaster and non-disaster tweets

Having decided to keep both the **text** and **keyword** columns as features, both of them were explored and preprocessed for both training and test set. The techniques used are the standard for NLP tasks: punctuation removal and special characters removal. Other standard techniques, such as

stopword removal and tokenization, were performed directly through two tools from the scikit-learn library: **CountVectorizer** and **TfIdfVectorizer**.

CountVectorizer is a feature extraction tool for NLP which automatically converts a collection of text documents to a matrix of token counts. This basic approach to text representation is commonly called "Bag of Words". CountVectorizer() also generates n-grams; in this case, I chose to use only unigrams, bigrams and trigrams, as tweets are usually not very long sentences. While it is faster than TF-IDF based methods, this type of vectorization tends to lead to worse algorithm performance, as will be shown in Section 4. TfIdfVectorizer is a feature extraction tool for NLP which performs the same job as CountVectorizer, but then converts the matrix of token counts to a matrix of TF-IDF features. TF-IDF stands for "term frequency - inverse document frequency" and is a typical NLP technique for evaluating the importance of a word both in its document and in the whole corpus. Again, I chose to use only unigrams, bigrams and trigrams. Both vectorizers were designed differently for text and keyword columns in order for them to be better tuned to each corpus' characteristic and vocabulary (See Figure 2).

3 Methods

Once the data had been preprocessed, five different machine learning models were trained on the dataset: **Logistic Regression**, **K-Nearest-Neighbours**, **Neural Networks**, **SVC** (Support Vector Machine for classification) and **Random Forest**. The decision not to try a Decision Tree algorithm was based on its poor performance with sparse data such as the preprocessed datasets for both CountVectorizer and TfIdfVectorizer preprocessing methods.

All models were trained using grid-search with 10-fold cross-validation for finding the best parameters. For **K-Nearest-Neighbours**, the optimal number of neighbours was found by first plotting accuracy and MSE for values of K between 1 and 60 and then observing the plots, yielding a different optimal K for different vectorizations of the data. For **SVC** (Support Vector Machine for classification), I used a LinearSVC (a faster version of SVC using a linear kernel) in the model applied to data vectorized with TfIdfVectorizer, as this data matrix's dimensions made the application of SVC with non-linear kernels rather unpratical (runtime over 1 hour).

For **Neural Networks**, different architectures were tried in a preliminary phase. Unsuprisingly, I found that feed-forward NNs reach best performance with 1 hidden layer, in line with the generally accepted result that a neural network with one hidden neuron layer is already capable of approximating arbitrary functions with any accuracy (Kriesel, 2007). Moreover, different optimizers (SGD, SGD with Nesterov's momentum, Adam, Adagrad) and activation functions for the hidden layers were tried in this initial phase. I used binary crossentropy as a loss function and a single output neuron with a sigmoid activation fuction for the output layer, as is standard for binary classification problems like the one at hand. Moreover, I implemented an early stopping procedure which stops training after 50 epochs with no performance improvement. The possibility of different architectures for different vectorizations of the data was explored, leading to the choice of doubling the number of neurons in the hidden layer for the BoW vectorization, but not for the tf-idf based one. For the hidden layers, I used both a kernel weight initializer in order to avoid the exploding/vanishing gradient problem and a kernel regularizer to reduce overfitting (HeNormal - the standard for ReLu activation function - and l1-regularizer with l1=0.0005, respectively).

A summary of the parameters used in the final implementation of the models can be found in Figure 3.

Alternative approaches include both different vectorization techniques, such as Word2Vec, the state-of-the-art for NLP, and different models, such as Recurrent Neural Networks, which are the state-of-the-art for NLP tasks, but they were excluded due to time and expertise constraints.

4 Experiments

Experiments were performed on the training set, split in validation and training sets with a 70/30 split. Part of the selection of the parameters was done empirically, such as the choice of activation functions and learning rates for neural networks, yet most of the parameters, as stated in Section 2, were found through grid-search 10-fold cross validation (see Figure 3 for best parameters of each model, highlighted in bold).

Performance evaluation was done by jointly observing balanced accuracy, confusion matrix and f-1 score for each of the (best) models trained. **Balanced accuracy** (BA) is a variation of the accuracy score, which , if the classifier performs equally well on either class, reduces to conventional accuracy. In contrast, if the conventional accuracy is higher only because the classifier "takes advantage" of an imbalanced test set, then the balanced accuracy, as appropriate, will drop (see Equation 1 below). It was used in order to avoid uncorrect estimation of accuracy due to the slight imbalance of the classes in the training set, highlighted in Section 2. In the case of neural networks, standard accuracy was used for simplicity.

$$BA = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \quad (1)$$

Confusion matrices and f-1 score were useful to determine the precision and recall levels of the various models. Most of the experimentation was actually done in the preprocessing phase, trying different models on different vectorizations on the data and seeing which combinations worked best. Overall, the TfIdfVectorizer seemed to provide better results for most models, as would be expected.

After experimenting with all the models, the most promising ones resulted to be the ones applied on data vectorized

Vectorizer Object / Parameters	ngram_range (lower and upper boundary of range of n-values for n-grams to be extracted)	max_features (build a vocabulary only considering the top-x features ordered by term frequency in the corpus)	min_df (ignore terms with document frequency lower than the given threshold when building vocab.)	stop_words (list of stopwords to be removed)
CountVectorizer for 'text'	(1,3)	200	6	'english'
CountVectorizer for 'keyword'	(1,3)	10	2	'english'
TfIdfVectorizer for 'text'	(1,3)	2000	10	'english'
TfIdfVectorizer for 'keyword'	(1,3)	500	3	'english'

Figure 2: Summary of parameters used for text vectorization

Model	Parameters	Values
Logistic Regression (CountVec)	C fit_intercept	[0.1,0.5,**1**,5,10,15,20] [**True**, False]
Logistic Regression (TfIdf)	C fit_intercept	[0.1,**0.5**,1,5,10,15,20] [**True**, False]
SVC (CountVec)	C kernel	[**1**,3,5,10,15] [linear, **rbf**]
LinearSVC (TfIdf)	C fit_intercept	[**1**,3,5,10,15] [**True**, False]
Neural Network	n. hidden layers optimizer activation function in hidden layers learning rate	[0,**1**,2,3] [SGD, **SGD+Nesterov's momentum(0.0005)**, Adagrad, Adam] [**relu**, softmax, elu, exponential] [0.001,0.002,0.003,0.006,0.008,0.009,0.01,**0.015**,0.2,0.3]
KNN (TfIdf)	K	[1,2,...,**13**,...,60]]
Random Forest (CountVec)	n_estimators criterion max_depth	[1,5,10,15,17,19,20,**21**,22,25] [gini, **entropy**] [None,1,3,5,10,**50**]
Random Forest (TfIdf)	n_estimators criterion max_depth	[1,5,10,15,17,19,20,**21**,22,25] [**gini**, entropy] [**None**,1,3,5,10,50]

Figure 3: Parameters tried manually and during grid-search for each model. Best parameters are highlighted in bold

with TfIdfVectorizer, in particular Logistic Regression, LinearSVC and Neural Network.

Model	Accuracy	F-1
Logistic Regression (CountVec)	**78.7**	73.8
Logistic Regression (TfIdf)	74.6	63.8
Neural Network (TfIdf, on whole training set)	76.1	/
SVC (CountVec)	75.8	66.0
LinearSVC (TfIdf)	**77.6**	73.0
KNN (TfIdf)	73.5	72.3
Random Forest (CountVec)	75.9	60.8
Random Forest (TfIdf)	75.6	70.7

Figure 4: Accuracies and F-1 scores on validation set for best models.

It is worthy to notice that great care was spent in experimenting different parameter settings and architectures for the neural network, but no clear improvement was obtained: while it obtained a validation accuracy over that of Logistic Regression (79%) when it was trained on a smaller set of data, this metric decreased to 76% once the network was trained on the whole training set (validation split=0.2). I believe this is simply an inevitable fault of feed-forward neural networks and much better performance could have been obtained using Recurrent Neural Networks, which are indeed the state-of-the-art for NLP tasks.

Validation accuracies and F-1 scores (when calculated) for all best models are reported in Figure 4. Figure 5 shows the confusion matrix of Logistic Regression on the validation set (as for the test set, true class labels were unavailable). We can observe that the algorithm correctly classifies 682 *real* disaster tweets out of 971, while it misclassifies 193 *fake* ones.

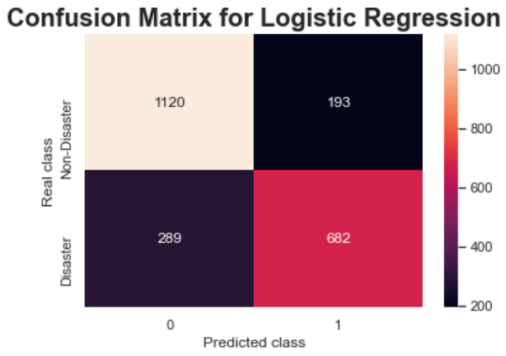

Figure 5: Confusion matrix for best model (Logistic Regression)

As highlighted in Figure 4, best validation set accuracies are obtained through Logistic Regression and LinearSVC. Training the best Logistic Regression model on the test set and sumbtting the prediction to Kaggle yields a test-set F-1 score of 77.9%.

5 References

David Kriesel, 2007, A Brief Introduction to Neural Networks, available at http://www.dkriesel.com