disastAIr: Evaluating Relative Efficiency of ML Models for Classification of Disaster Tweets using NLP

Maria P. Jimenez M., Lennart K.M. Schulz, Laura I.M. Stampf, Dovydas Vadišius
Vrije Universiteit Amsterdam

Abstract

This paper aims to identify the footprint cost associated with increases in the performance of machine learning models, a metric that is often overlooked in related work. Yet, this metric is potentially important in the scalability evaluations of model deployment or even the overall economic and environmental feasibility of a deployment. For this, a classical machine learning model (Markov model) is compared to a deep learning model (BERT) for the task of classifying disaster Tweets using natural language processing. First, the performance of both models is evaluated to identify whether they could realistically be used in production. Subsequently, the models are compared on their footprint cost for both classification and training. The experiments showed that, while the deep learning model outperformed the classical model, this increase in performance came with a significant increase in footprint cost. These findings may suggest that applications, where machine learning models were previously deemed infeasible due to their footprint cost, could be satisfied by simpler models with a lower footprint cost at only a small performance penalty

Keywords: *Footprint Cost, Benchmarking ML Models, Natural Disaster Management, Disaster Tweets, Deep Learning, Markov Models*

1 Introduction

Natural disasters (NDs) have had and continue to have tragic effects on society, resulting in the loss of thousands of lives annually. However, the past decade has witnessed a significant reduction in the yearly death toll due to NDs, with a decrease of more than 92% from 1920 to 2020 [1]. According to Our World in Data [1], the significant reduction in deaths can be attributed to many factors including the improvement in infrastructure and emergency medical services.

Disaster management (DM) has become a crucial component in lessening potential effects caused by NDs. The use of technology to aid in DM has become increasingly popular [2]. Given that social media is the most used data source to gather information about disasters, especially Twitter (now called X) [3], there is immense potential to leverage such platforms for further development in disaster management, as is already being done by numerous people (see Section 2). An example application could be the monitoring of Twitter feeds while identifying each Tweet for whether it gives information about a disaster. Such Tweets could then be forwarded to the appropriate disaster response entity, possibly fastening the response times by providing valuable information.

To make the natural language used in social media usable for machines, some means of natural language translation must occur. Natural language processing (NLP) does just this; it is "devoted to making computers understand the statements or words written in human languages" [4]. In regards to disaster management, the focus of NLP is more aimed at natural language understanding, or linguistics, as opposed to natural language generation.

NLP has been an area of machine learning (ML) research for many decades. In the 1990s, Markov models were often used for language-related tasks due to their simplicity and computational efficiency [5, 6]. However, more recently, larger, so-called "deep learning" (DL), models have become increasingly popular. While Markov models are still used in the domain, their use nowadays is mostly in areas like *Named-Entity Recognition* and *Parts of Speech Tagging* [7].

Generally, DL models seem to upend the performance of classical models in, among others, classification tasks [8]. It is, however, unclear how this improvement relates to the increased footprint cost of the computationally more intensive deep models, a question often not considered in similar work.

This paper aims to investigate the potential use of NLP for disaster management using Tweets. The performance in terms of correct natural disaster detection will be compared across a classical and a DL model. With scalability, economic feasibility, and sustainability in mind, the models will be further assessed on their footprint cost in terms of computational load.

1.1 Research Questions

Research Question 1 (RQ1):
Can a classical ("simple") machine learning model predict whether a Tweet contains information about real disasters with a high enough performance such that it could be used in practice (e.g., in a disaster management system)?

Research Question 2 (RQ2):
Is a deep learning model able to outperform the classical machine learning model (from RQ1)? If so, how large is the improvement?

Research Question 3 (RQ3):
What computational performance cost does the deep learning model have in comparison to the classical model (both for training and for predictions)? How does this computational cost relate

to the performance of the model?

2 Related Work

Natural Language Processing has been widely used to analyze social media Tweets for different purposes, including sentiment analysis to predict financial trends [9] or to analyze COVID-19 vaccination sentiment [10], to detect cyberbullying [11], offensive language [12], healthcare symptoms and diseases [13], and detecting suicidality [14].

Analysis of Tweets with NLP has also been employed for detecting disaster-related Tweets. In [15], the authors identify disaster-related Tweets with fine-grained details of "affected individuals, damaged infrastructure and disrupted services", and they include an evaluation of Tweets that allows to differentiate between space and time periods when extracting information about disasters. The data was collected and labeled by the team of researchers from Tweets generated before, during, and after Hurricane Irma, and the total size of the dataset (after filtering) was approximately 550,000 Tweets. The text was pre-processed by removing punctuation and URLs, transforming all characters to lower-case, tokenizing into words and transforming the vector Tweet into a set of word frequencies. The team of researchers experimented with various models including Logistic Regression, Linear SVMs and Ridge, CNNs and LSTM Networks. It was concluded that LSTM networks surpass in performance for this task of binary classification, with an F1 Score of 0.75.

In their paper, Alvarado *et al.*[16] pre-processed publicly available data of approximately 10,000 Tweets and chose as classifier RoBERTa, to then create an emergency chatbot suited to determine if an input text is a real emergency situation or not. The selected classifier was compared against the Multinominal Naive-Bayes classifier and was preferred given its higher accuracy. In contrast to Sit *et al.* [15], the authors did additionally include stemming, lemmatization, and POS tagging as part of pre-processing the text.

Bikku *et al.* [17] performed an analysis of disaster Tweets and text mining on a dataset of 10,000 Tweets. Similarly to [15], the Tweets were pre-processed by removing punctuation and stop-words, doing tokenization and stemming. The authors employed MPP, KNN, Random Forest, Optimized SVM, K-means, and BPNN. The highest accuracy was achieved with optimized SVM, even though when compared to all the other models it had lower individual values for precision, recall, and F1 Score.

ML models and more specifically DL models have evolved rapidly throughout the years and this has led to an exponential increase in the training data and model capacity [18], an increase in the network complexity, number of parameters, amount of training resources, among other things [19].

It is worth noting that even though these models might achieve high performance for the task they are trained for, this might not necessarily entail that the model is efficient enough to be deployed in a production environment [19] nor that the research community takes this aspect into account when reporting results and comparing different models ([15], [16], [17]).

According to Menghani [19], there exist two types of related metrics under which a model's efficiency can be measured. On the one hand, there are quality metrics that refer to the capability of the model to accomplish the task, such as accuracy, precision, recall, and F1 Score. On the other hand, there are footprint metrics that are associated with the cost of deploying a model. This group of metrics includes the number of parameters, RAM consumption (peak and average), model disk size, number of epochs to converge, and latency, among others.

Taking into account the potential scalability of these models (and as it is highlighted by [19]), to compare the relative efficiency between two different models, it is crucial to contrast both quality and footprint metrics.

Instead of the performance optimization objective, it would be more relevant to achieve Pareto-optimality, where a Pareto-optimal model is the one that is not dominated by any other feasible solution [20]. In this case, that would mean a model where no metric can be improved without sacrificing another. So a model should be chosen as the best for the trade-off between model quality and model footprint (depending on the relative costs).

All of these considered, NLP has been used to detect disaster-related Tweets by using several classical ML models and DL strategies. The size and origin of the data vary and the pre-processing of the Tweets depends from study to study. To have a more realistic comparison between the efficiencies of the two models, quality and footprint metrics should be equally considered. In contrast to other research performed within NLP to detect disaster-related Tweets, in this paper, we aim to identify the trade-off between performance and footprint cost of machine learning models.

3 Data & Pre-processing

3.1 Dataset

I Kaggle Dataset

The dataset for this paper was obtained from a Kaggle competition [21] that incites teams to predict which Tweets are about real disasters and which ones are not. The data consists of 7,613 instances used for training and 3,263 instances used for testing. The features of each instance include the text of a Tweet (string of characters), a single keyword from that Tweet and the location the Tweet was sent from. The latter two features might be blank and also not expected in production, hence we did not consider them to make predictions in our models.

II Labeled Test Set

The test set given as part of the Kaggle competition does not include labels. Results of the classification can only be obtained by uploading the predicted labels to the competition and observing the received F1 Score. This, however, does not allow for detailed analysis of various result metrics (e.g., precision, recall) or for the automation of the test process.

Investigations into the origin of the given dataset, while not yielding any specific source, brought up a GitHub repository [22] containing a seemingly similar dataset, however, with more detailed labels (e.g., `choose_one:confidence`, likely indicating the confidence for the given label). After some issues with character encoding discrepancies, we were able to match the instances from the Kaggle test set to instances in this labeled set.

In order to check if this newly found labeled set corresponded appropriately to the Kaggle test set, the former was submitted as a "prediction" to the competition and achieved an F1 Score of ∼0.949, indicating that, while the majority of labels were correct, there were still some instances that did not match the expected label. Further examination of the labeled dataset revealed that some (∼20) instances appeared multiple times, yet with varying labels, and some instances even had the label "Can't Decide". As these instances were likely the ones incorrectly labeled in the submitted prediction, they were modified one-by-one, checking the new Kaggle F1 Score for each change.

Ultimately, the F1 Score reached ∼0.998, at which point it was decided to stop spending time on this process and work with this (almost fully correctly labeled) test set for the final evaluation of the models.

3.2 Data Pre-processing

I Class Imbalance

Class imbalance can have a significant effect on model validity. Imbalanced data is commonly mitigated through over-sampling the minority class or under-sampling the majority class. It is important to investigate whether a class imbalance is present in the disaster Tweet data to assess where the training data should be adjusted. To test for class imbalance, the number of instances present for each target class (Positive or Negative) is plotted in Figure 1.

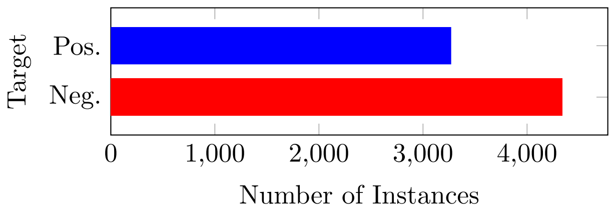

Figure 1: Histogram for Class Imbalance

The histogram in Figure 1 helps to identify that there are more negative than positive instances in the dataset. Normalizing the counts for both classes reveals that the majority class (negative) contributes approximately 57.03% to the total data set, leaving 42.97% positive instances. Though a slight discrepancy of 7.03% from the 50:50 split exists between the number of class instances, the difference is not considered significant enough to warrant over-sampling or under-sampling, as these would risk over-fitting or information loss, respectively.

Even though the class imbalance within the training and test datasets appears to be minor, it is likely that this imbalance is considerably more pronounced in a real-world production environment, given the vast quantity of non-disaster-related Tweets that are typically present on the platform.

II Cost Imbalance

Careful consideration of the cost imbalance is important for a thorough evaluation of any model's performance. In the context of this project, false negatives (FN) represent disaster Tweets that are misclassified as non-disaster Tweets. False positives (FP) represent Tweets that are not about a disaster but are classified as such. In the scope of this project, the preference of FNs or FPs is heavily dependent on the application of the model. For instance, if a disaster Tweet classifier were to be used to dispatch ambulances to disaster scenes, it would be inconvenient and costly to wrongfully respond to FPs. On the other hand, it could be life-threatening not to dispatch an ambulance to a scene where a disaster is happening but was not classified correctly, as is the case for FNs. It is advised that the classification results of these models are not taken at face value - human input to determine the validity of results is advised if models are to be used to alert of real-world disasters.

4 Method

4.1 Classical Model (Markov)

Classical machine learning classifiers can typically be grouped into discriminative and generative classifiers. The choice between the two may generally depend on many factors, including the expected distribution of the model underlying the dataset [23]. While it has been shown that the asymptotic error for discriminative classifiers is lower than that of their generative counterpart, this level of performance is typically only reached with a sufficiently large set of training data [24]. As the size of the dataset for this research is rather small, a generative classifier is likely more appropriate. Furthermore, generative classifiers are one of the simplest classifiers to build and train.

Thereby, a (generative) Markov model classifier was chosen as the classical ("simple") ML model for this research.

I Generative Classifiers

A generative model, like the Markov model, learns the distribution of data in the feature space. Through this learned distribution, the model can then predict how likely it is for some given data to be coming from the same distribution. This idea can easily be turned into a classifier by learning multiple instances of the same model, each on the data labeled with a certain class. By applying Bayes' rule, the conditional probabilities of some data x being generated by the distribution of a class Y can be used to extract the probability for a class Y conditioned on the data x. Once this conditional probability has been determined for all classes, the class with the highest probability will be used as the prediction for the given instance.

II Markov Assumption

The intuitive approach for estimating the probability of a given sentence s is to take the relative frequency of its occurrences in the data. Such an approach, however, would (aside from requiring an infeasibly large dataset) not be useful for estimating probabilities of new sentences that have never been seen before. To mitigate this issue, multiple steps are taken. Firstly, the sentence is broken up into tokens $t_1, ..., t_k$ (which can be on the character level, word level, or something in between), and the probability of the sentence is estimated by the joint probability of all the tokens in the sentence:

$$p(s) = p(t_1, ..., t_k) \tag{1}$$

This joint probability can be rewritten to the product of the probability of each word conditioned on all the words that preceded it:

$$p(s) = \prod_{i=1}^{k} p(t_i | t_1, ..., t_{i-1}) \tag{2}$$

through the chain rule of probability. As this still requires estimations of conditional probabilities for unfeasibly many conditions (i.e., all preceding words in a sentence), the Markov assumption simplifies the condition by assuming that each word only depends on the n words before it (called an n-gram). These conditional probabilities can then easily be estimated by the frequency of the given sequence of $n+1$ tokens relative to the frequency of the sequence of the first n tokens[1]:

$$p(t_{n+1} | t_1, ..., t_n) \approx \frac{\#t_1, ..., t_n, t_{n+1}}{\#t_1, ..., t_n} \tag{3}$$

The model based on this assumption is called a Markov model, where n is a hyperparameter, called the *order* of the model. With Equation 3, n-grams that did not appear in the data will be assigned a probability of 0, consequently giving the whole sentence a probability of 0. To avoid this, Laplace smoothing can be applied, adding so-called pseudo-observations for each possible n-gram:

$$p(t_{k+1} | t_1, ..., t_k) \approx \frac{\#t_1, ..., t_k, t_{k+1} + \alpha}{\#t_1, ..., t_k + \alpha(|\Sigma|^k)} \tag{4}$$

where Σ is the alphabet of all possible tokens[2] and α is the smoothing factor.

III Implementation

To our knowledge, no existing Python implementation of a Markov Classifier for word or character sequences exists. While libraries like `nltk` offer related features, the implementation details and with that the possible performance overheads due to additional features or generalized implementations are not known, possibly skewing the results relevant for RQ3. Thereby, a simple implementation of a Markov Classifier that follows the details explained in Section 4.1 and can work with sequences of any kind (character and word level tokens) has been written as part of this research. The source code can be found, along with all other project data, on GitHub[3].

The implementation focuses on simplicity, to not compromise performance, while still offering an API that includes functions for many features that would be needed in a production setting. Furthermore, the training process is implemented in both a single-threaded and a multi-threaded way, with the latter implementation drastically reducing the training times for large datasets. For smaller datasets (like the one used for this research), the added overhead associated with the setup of the thread pool outweighs the performance gains and thereby makes the single-threaded version favorable.

IV Model Selection / Hyperparameter Exploration

As noted before, sentences can be broken down into various kinds of tokens. The most intuitive and simple (and with that interesting for this intentionally simple model) tokenization techniques are character-based tokens and word-based tokens. While a word-based model seems to be more intuitively appropriate for NLP, it comes at the cost of requiring larger training sets (as there are many more unique tokens and with that more unique n-grams), possibly requiring more extensive pre-processing (e.g., through stemming), and not being able to handle typos or alternative spellings well.

Another consideration is the order of the model. A 0th-order model, while cheap to train, only uses the absolute frequencies of all words of the sentence (thereby representing a naive Bayes classifier). Higher orders can use more information about the structure of the sentence. However, also higher orders increase the number of unique n-grams exponentially with respect to the vocabulary size, again requiring larger datasets to allow for decent estimates of the probabilities.

For this research, 3 different types of tokenization were considered: character-based without any pre-processing, character-based with all characters converted to lowercase, and word-based with all charac-

[1]This clearly does not work for the first token of a sequence. In that case, the probability is simply estimated by the number of occurrences of the token relative to the total number of occurrences of all tokens.

[2]In practice, an approximation of the total number of common unique tokens is used - further discussion of these techniques, while interesting (e.g., [25]), goes beyond this paper.

[3]`https://github.com/lkm-schulz/disastAIr`

ters converted to lowercase and link texts removed (mostly following the approach highlighted by Sit *et al.* [15]). For each of these types, 16 different orders (0 to 15) were considered. While higher orders are possible, initial tests showed that no model improved in performance for orders 8 or higher on the given dataset.

Due to the relatively small size of the training dataset, cross-validation was used for the hyperparameter exploration and evaluation process. The training data was split into 5 chunks, each chunk being used once as a validation set (with the remaining 4 chunks being used as training data). The average of the results of each split was used as the overall result.

4.2 Deep Learning Model (BERT)

The Bidirectional Encode Representations from Transformers (BERT) model was chosen as the DL model to use, a powerful sequence model built on sequence-to-sequence layers known as self-attention. Self-attention has two main features: its ability to perform parallel processing and use a virtually infinite memory. Using self-attention, inputs fed into the sequence-to-sequence layer are transformed into a weighted sum of outputs. A notable difference that contributes to the power of the BERT model is that these weights are derived from its inputs rather than provided as user-set parameters. Additionally, as implied by the "B" in its name, the BERT model works bidirectionally, allowing the model to capture the context of a token in both the left and right directions at once. BERT's reputable performance as a language model is also attributable to the fact that it is first pre-trained in an unsupervised way before continuing to finetune the labeled data.

Specifically, the paper's implementation of the DL model uses DistilBERT, a smaller and faster version of the BERT model that helps to alleviate the time and computational constraints of the project. By removing token-type embeddings, and reducing the number of layers by a factor of 2, DistilBERT is 60% faster when compared to the BERT model, and it retains 97% of language understanding capabilities [26].

I Implementation

The implementation uses an existing pre-trained DistilBERT model from `keras_nlp` library, with two possible classes, and a maximum preset length of a message of 160 characters (maximum length of a Tweet in the dataset). Then, the model is trained on the training data using Adam optimizer [27], with a learning rate of 10^{-5}, and two hyperparameters: batch size for a training iteration and epochs (number of iterations through the dataset).

II Hyperparameter Exploration

As the paper aims to find the impact of batch size and epoch count on the F1 Score, a grid search was conducted to find the best hyperparameter values, where epochs varied between 1 and 16, while the considered batch sizes were integer powers of 2 from 1 to 512. As the DistilBERT fine-tuning takes a significant amount of time, each combination of batch size and epoch count was tested once, using the last 20% of training set as the validation set for hyperparameter selection. Larger computational power and time resources would allow to choose bigger values of the hyperparameters and run counts in future work.

4.3 Evaluation

To establish a baseline measurement, giving a context for the results of the models, 3 different classifiers were used: The average of 1,000 runs of a random classifier (assigning positive and negative labels at random), a majority-class classifier, and a majority-class classifier.

Even though there is not a significant class imbalance in the training data, potentially there would be in production setups, we use the F1 Score over the accuracy of the models to be able to give more accurate interpretations of the model performance.

I Evaluation Metric: F1 Score

Model performance is evaluated using the F1 Score. F1 Scores represent the "harmonic" mean of precision and recall by giving equal weights to both metrics. The tradeoff between precision and recall is an important factor in model evaluation; precision is important when the goal is to decrease the number of FPs, while recall is good for a low number of FNs. In the context of this project, there is no prioritization of FPs and FNs as this depends heavily on the production use of a model (see Section 3.2-II about Cost Imbalance). Thus, using an F1 Score where both precision and recall contribute equally makes it an appropriate metric for model evaluation.

4.4 Footprint Cost Benchmarking

Footprint costs are generally composed of many individual factors (as highlighted in Section 2). When evaluating whether a model would be economically feasible to be used in production, the most dominant factor is likely the CPU time (which is the time of active CPU usage). Not only does CPU time give a good estimation of overall computational load and thereby energy consumption, but CPU time is furthermore used as the prevalent metric for billing of cloud-hosted compute resources [28]. Thereby, CPU time can often be directly related to the overall cost of deploying a system.

4.5 Testing Environment

It is important to ensure a controlled environment when benchmarking different algorithms to lessen the effect of extraneous factors. Since a large component of this paper is the comparison of two models based on their computational performance (as stated in RQ3), the models are run on the DAS-5[4], a distributed ASCI supercomputer with six clusters, one of which is hosted at the Vrije Universiteit Amsterdam with 68 nodes. The DAS-5 is engineered for research, being a lot less susceptible to random noise in its measurements [29]. By testing both models on

[4]`https://www.cs.vu.nl/das5/`

Type	Accuracy	F1 Score
Random	0.500	0.461
Minor	0.428	0.600
Major	0.572	0.000

Table 1: Baseline Accuracy and F1 Scores

the DAS-5, a more controlled environment than a personal computer is achieved.

Another advantage of the DAS-5 is that each node, consisting of 16 cores, can run different processes simultaneously without interference. This is useful for running a model with different hyperparameters in parallel. Deep learning models, such as the BERT model used in this project, are computationally intense and require a significant amount of time to complete training (less so for prediction), thus, access to the DAS-5 significantly lessened the time needed for hyperparameter optimization, reducing the time scale by an entire magnitude of 10 (from approx. 100 hours to 10 hours).

5 Results & Discussion

5.1 Baselines

Table 1 shows the accuracy and F1 Score for each of the 3 baseline classifiers. The results are as expected: The random classifier gets exactly 50% right, while the majority and minority class classifiers reflect the distribution of positive and negative instances in the data. The majority class classifier furthermore has an F1 Score of 0, as it does not classify anything as positive.

5.2 Classical Model (Markov)

Figure 2 shows the F1 Scores of the 3 different models for orders 0 to 15 on the cross-validation sets (as explained in Section 4.1-IV). This data exhibits multiple interesting characteristics of the different types of Markov models.

The word-based model has the highest maximum F1 Score of ∼0.75, indicating that word-level tokenization might be more expressive. This is expected, as (sequences of) words convey more meaning than (equally long sequences of) characters in natural language. However, the F1 Score of the word-based model is only high for order 0 (so essentially a naive Bayes' classifier) and drops significantly with increases in the order of the model. As explained in Section 4.1, increasing the order of the model drastically increases the number of unique n-grams, and with that the size of the training data needed for good estimations of their probabilities. The training set used here is seemingly too small for a word-based Markov classifier of higher orders.

Both character-based models exhibit a similar behavior where the performance is comparatively bad for low orders, reaches a peak at order 6 or 7, and then starts dropping off slightly. Evidently, simple single-character frequencies are not useful to estimate the class of a given text. For orders around 6, the

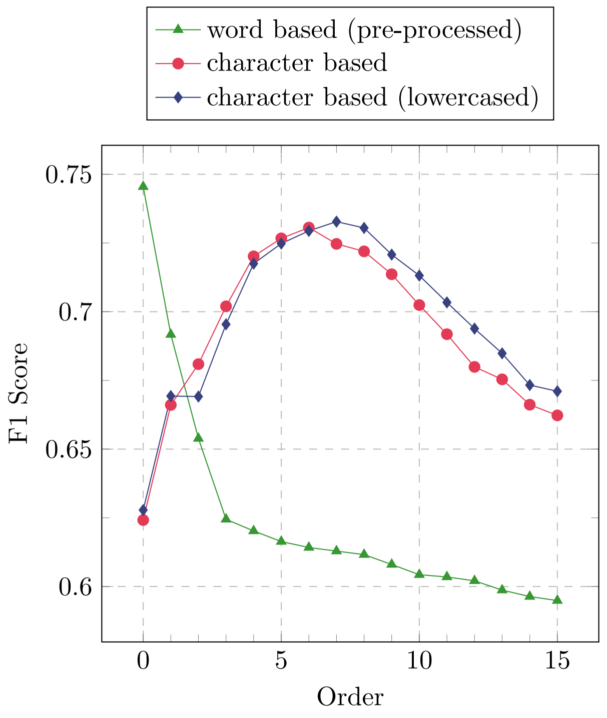

Figure 2: F1 Scores of the Markov Model classifiers on a (cross-)validation set.

sequences are long enough for the distributions of the two classes to be sufficiently different such that the predictions reach an F1 Score of ∼0.73. For higher orders, also the character-based models seem to suffer from the same issue as the word-based model: the dataset is not large enough to provide good estimations of the frequencies of such long n-grams. The model using all lowercase characters works better for higher orders than the other character-based model. This is unsurprising, as there are less unique n-grams with an alphabet of only lowercase letters compared to the full alphabet.

From these results on the validation set, clearly, the 0th-order word-based Markov classifier is the most preferable one. Figure 3a shows the performance of this model on the test set, with an F1 Score of ∼0.739.

5.3 Deep Learning Model (BERT)

As explained in Section 4.2-II, a grid search was performed to find the combination of hyperparameters for the DistilBERT model that exhibits the best performance in terms of F1 Score.

Figure 4 shows the F1 Scores of the deep learning model on a validation set. It can be seen, that there is no direct relation between batch size and performance or number of epochs and performance. Rather, different choices for the number of epochs exhibit different characteristics in the relation between the batch size and the F1 Score.

The models trained through 1 and 2 epochs show a clear drop in performance for larger batch sizes. This drop is likely due to the consequently low number of total gradient descent steps that are taken during training. For instance, for 1 epoch with a batch size

		Pred.	
		Pos.	Neg.
Act.	Pos.	989	409
	Neg.	291	1,574

(a) Markov, 0th, word, pre-processed

		Pred.	
		Pos.	Neg.
Act.	Pos.	1,099	299
	Neg.	281	1,584

(b) BERT, 4 epochs, 64 batch size

Figure 3: Confusion matrices for the predictions on the test set of the selected Markov and BERT classifiers.

of 256 elements, only ∼30 steps of gradient descent are taken with the given training set size, while for 16 epochs with the same batch size, ∼360 steps of gradient descent are taken.

For higher epoch counts, increases in the batch size also tend to increase the performance of the model.

When sorted based on their performance for very small batch sizes, the ordering of the models seems to be arbitrary. For larger batch sizes, most notably 512, the same method of sorting puts the models in a strictly increasing order of the number of epochs, indicating that for larger batch sizes more epochs give rise to better models.

Of all the DistilBERT model variations, the one trained with a batch size of 64 and 4 epochs is the most promising. Figure 3b shows the performance of this model on the test set, giving an F1 Score of ∼0.784.

For the analysis of the fingerprint cost, also the models with 1, 2, and 16 epochs are considered, each with the batch size that was the most promising based on the validation results. Notably, however, these models are not considered in the discussion of RQ2 and RQ3, as that could possibly be seen as an error of performing multiple testing on the test set.

5.4 Footprint Cost

Figure 5 plots the selected models in the space of F1 Score (for the test set) and average classification time per instance. This plot nicely highlights the trade-off between computational load and classification performance. The 6th- and 7th-order Markov models have an F1 Score of ∼0.73 with an average classification time in the high hundreds of microseconds. The DL models, on the other hand, manage to improve the F1 Score by ∼0.06, however, this improvement comes at the cost of an average classification time that is more than 2 orders of magnitudes higher. Interestingly, however, the word-based Markov model is also able to achieve a higher F1 Score, compared to the other Markov models, yet, its average classification

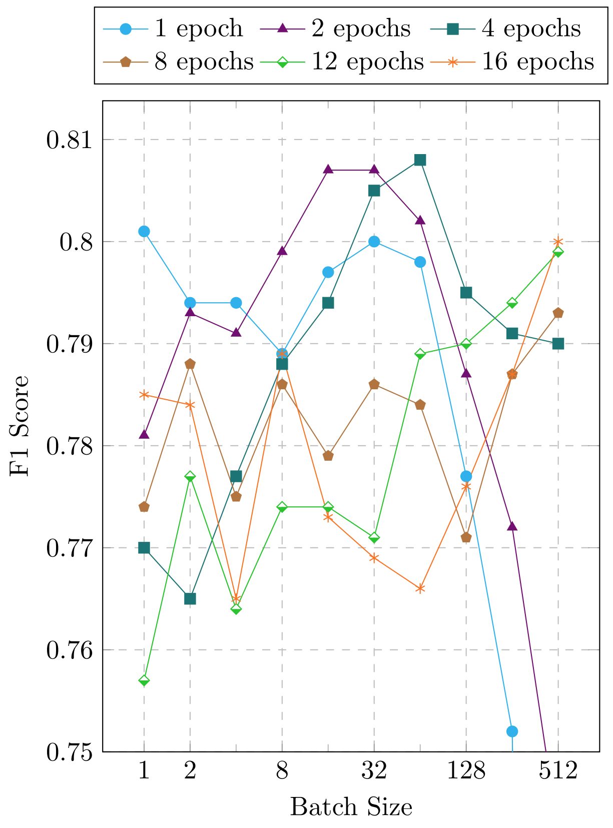

Figure 4: Deep learning model validation set results

time is almost one order of magnitude lower. Improvements in F1 Score seemingly do not necessarily entail higher footprint costs in terms of classification times. A more fitting model, like a word-based classifier instead of a character-based classifier, can yield better performance at a lower cost, highlighting the importance of a careful choice of the model.

In order to compare any two models for their relative efficiency it is important to compare both the performance of the model and its footprint costs. The latter encompasses both the training cost (see Figure 6) and the inference cost (Figure 5). The training time required per model follows the trend of the time it takes to predict an instance, that is, the simpler the model, the less time it takes to train.

It is worth noting, however, that even though the training time is taken into consideration, training is done potentially only once in the lifetime of a model, while predictions are made numerous times. Giving a higher priority to the metrics referring to a model when it is predicting.

6 Conclusion

The aim of this paper is to investigate the potential to use NLP to classify Tweets into disaster related content or not. In order to approach this task, we have focused our research into RQ1, RQ2, and RQ3.

Regarding RQ1, a classical ML model, in our case a 0th-order word-based Markov model, predicts whether a Tweet contains information about a real disaster with a performance that is significantly above

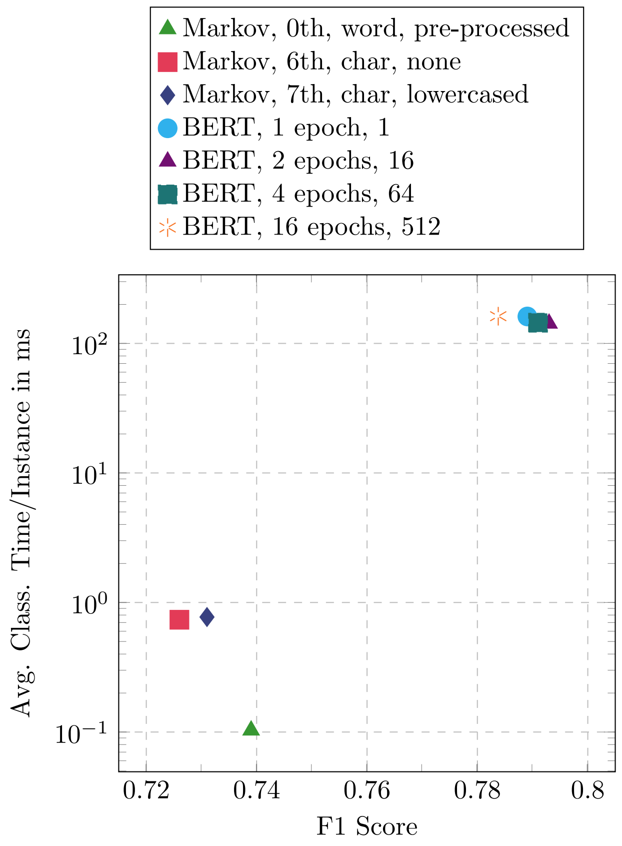

Figure 5: Scatterplot for F1 Score vs. average classification time per instance.

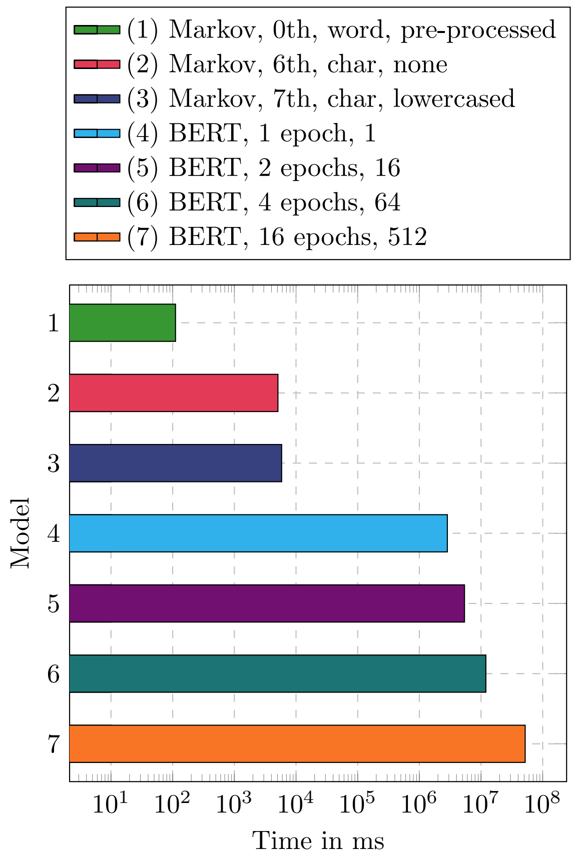

Figure 6: Histogram for Training Time on 7,613 instances

the baseline ($F1 \approx 0.74$). To determine if this model can be used in practice, depends mainly on the context in which it is used for. As mentioned in Section 3.2-II, it would likely be necessary to include human feedback to validate the predictions performed by the model before taking action.

Despite the fact that a classical ML model is able to achieve a high performance, a BERT model trained through 4 epochs with a batch size of 64 elements (the best among the deep learning models tested) outperforms the 0th-order word-based Markov model (RQ2) with an F1 Score improvement of ~0.06.

Reaching this slightly better accuracy with the deep learning model, however, comes at the cost of an increase in average classification time by 3 orders of magnitude and an increase of the training time of 5 orders of magnitude (RQ3). Therefore, if computational resources are abundant there would be an inclination to choose the deep learning BERT model over the, more simple, Markov model. If on the other hand, there is a limitation on computational resources (e.g., through the financial cost of computation) and minor decreases in performance are acceptable, the Markov model is an alternative option.

6.1 Limitations

Understanding the limitations of this paper is crucial for accurately interpreting its findings and implications. The following limitations were identified:

Sample Size: The dataset used for training consists of 7,613 instances, which is a rather low amount in the area of machine learning, especially in NLP. Nevertheless, the most notable limitation this poses to the paper's findings is regarding the word-based Markov model; the limited dictionary size that the model is trained on led to a steep decline in its performance for higher orders (see Figure 2). A larger dataset could potentially reveal increases in performance for higher orders similar to the ones exhibited by the character-based models.

Class (Im)Balance: The training dataset used has an insignificant class imbalance of 7.03% from a 50:50 split. However, as is briefly mentioned in Section 3.2-I, the balanced test data is likely not representative of a production setting. Hence, the deployment of these ML models in a production setting could result in significantly different performance.

Number of Models: The paper considers and compares two models: the Markov and BERT model. This highlights a limitation in the scope of the paper. The inclusion of further models could reveal more findings about the trade-off between fingerprint cost and model performance.

Time and Computational Resources: The average results (F1 Scores and times) gathered for the

DL model were taken over three runs per combination of epoch and batch size due to limitations in computational resources. This poses a limitation to the validity of the mean as a generalization of the model's performance. The mean may be heavily impacted by potential outliers in the raw data collected, hence misrepresenting the performance of the DL model.

6.2 Future Work

Considering that only 2 groups of models (Markov models and BERT models) were compared in this paper, future research could investigate the performance of further different types of models like LSTM networks or optimized SVMs, following work performed by Sit *et al.*[15] and Bikku *et al.*[17], respectively. In this exploration of models, it could be valuable to try to identify clusters of models in the space of F1 Score vs average classification time, so models that are similar both in performance and footprint cost. This could help to get a further understanding of the relation between the capability of a model to accomplish the task (F1 Score) and the cost of deploying the model and to possibly identify a class of Pareto-optimal models.

References

[1] Our World in Data, *Natural disasters*, https://ourworldindata.org/natural-disasters, Accessed: 2024-03-14, 2024.

[2] M. Krichen, M. S. Abdalzaher, M. Elwekeil and M. M. Fouda, "Managing natural disasters: An analysis of technological advancements, opportunities, and challenges", *Internet of Things and Cyber-Physical Systems*, vol. 4, pp. 99–109, 2024, ISSN: 2667-3452. DOI: 10.1016/j.iotcps.2023.09.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667345223000500.

[3] R. R. Arinta and E. Andi W.R., "Natural disaster application on big data and machine learning: A review", in *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, IEEE, Yogyakarta, Indonesia, 2019, pp. 249–254. DOI: 10.1109/ICITISEE48480.2019.9003984. [Online]. Available: https://ieeexplore.ieee.org/document/9003984.

[4] D. Khurana, A. Koli, K. Khatter *et al.*, "Natural language processing: State of the art, current trends and challenges", *Multimedia Tools and Applications*, vol. 82, pp. 3713–3744, 2023. DOI: 10.1007/s11042-022-13428-4. [Online]. Available: https://doi.org/10.1007/s11042-022-13428-4.

[5] M. Zissman, "Automatic language identification using gaussian mixture and hidden markov models", in *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1993, 399–402 vol.2. DOI: 10.1109/ICASSP.1993.319323.

[6] R.-H. Liang and M. Ouhyoung, "A sign language recognition system using hidden markov model and context sensitive search", VRST '96, pp. 59–66, 1996. DOI: 10.1145/3304181.3304194. [Online]. Available: https://doi.org/10.1145/3304181.3304194.

[7] T. Almutiri and F. Nadeem, "Markov models applications in natural language processing: A survey", *International Journal of Information Technology and Computer Science*, 2022. DOI: 10.5815/ijitcs.2022.02.01. [Online]. Available: https://api.semanticscholar.org/CorpusID:249971023.

[8] Q. Li *et al.*, "A survey on text classification: From traditional to deep learning", *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 2, Apr. 2022, ISSN: 2157-6904. DOI: 10.1145/3495162. [Online]. Available: https://doi.org/10.1145/3495162.

[9] A. Gupta and V. K. Tayal, "Analysis of twitter sentiment to predict financial trends", in *2023 International Conference on Artificial Intelligence and Smart Communication (AISC)*, 2023, pp. 1027–1031. DOI: 10.1109/AISC56616.2023.10085195.

[10] J. Ye, J. Hai, Z. Wang, C. Wei and J. Song, "Leveraging natural language processing and geospatial time series model to analyze COVID-19 vaccination sentiment dynamics on Tweets", *JAMIA Open*, vol. 6, no. 2, ooad023, Apr. 2023, ISSN: 2574-2531. DOI: 10.1093/jamiaopen/ooad023. eprint: https://academic.oup.com/jamiaopen/article-pdf/6/2/ooad023/49871619/ooad023.pdf. [Online]. Available: https://doi.org/10.1093/jamiaopen/ooad023.

[11] A. M. Alduailaj and A. Belghith, "Detecting arabic cyberbullying tweets using machine learning", *Machine Learning and Knowledge Extraction*, vol. 5, no. 1, pp. 29–42, 2023, ISSN: 2504-4990. DOI: 10.3390/make5010003. [Online]. Available: https://www.mdpi.com/2504-4990/5/1/3.

[12] M. Anand, K. B. Sahay, M. A. Ahmed, D. Sultan, R. R. Chandan and B. Singh, "Deep learning and natural language processing in computation for offensive language detection in online social networks by feature selection and ensemble classification techniques", *Theoretical Computer Science*, vol. 943, pp. 203–218, 2023, ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2022.06.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397522003887.

[13] S. Alotaibi, R. Mehmood, I. Katib, O. Rana and A. Albeshri, "Sehaa: A big data analytics tool for healthcare symptoms and diseases detection using twitter, apache spark, and machine learning", *Applied Sciences*, vol. 10, no. 4, 2020, ISSN:

2076-3417. DOI: 10.3390/app10041398. [Online]. Available: https://www.mdpi.com/2076-3417/10/4/1398.

[14] A. Abdulsalam, A. Alhothali and S. Al-Ghamdi, “Detecting suicidality in arabic tweets using machine learning and deep learning techniques”, *Arabian Journal for Science and Engineering*, pp. 1–14, 2024. DOI: 10.1007/s13369-024-08767-3.

[15] C. K. Muhammed Ali Sit and I. Demir, “Identifying disaster-related tweets and their semantic, spatial and temporal context using deep learning, natural language processing and spatial analysis: A case study of hurricane irma”, *International Journal of Digital Earth*, vol. 12, no. 11, pp. 1205–1229, 2019. DOI: 10.1080/17538947.2018.1563219. eprint: https://doi.org/10.1080/17538947.2018.1563219. [Online]. Available: https://doi.org/10.1080/17538947.2018.1563219.

[16] B. J. S. Alvarado and P. E. C. Solano, *Detecting disaster tweets using a natural language processing technique*, 2021.

[17] T. Bikku, P. Chandrika, A. Kanyadari, V. Prathima and B. B. Sai, “Analysis of disaster tweets using natural language processing”, in *Intelligent Computing and Applications: Proceedings of ICDIC 2020*, Springer, 2022, pp. 491–501. DOI: 10.1007/978-981-19-4162-7_46.

[18] C.-J. Wu *et al.*, “Sustainable ai: Environmental implications, challenges and opportunities”, *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022. DOI: https://doi.org/10.48550/arXiv.2111.00364.

[19] G. Menghani, “Efficient deep learning: A survey on making deep learning models smaller, faster, and better”, *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023. DOI: 10.48550/arXiv.2106.08962.

[20] Y. Jin and B. Sendhoff, “Pareto-based multiobjective machine learning: An overview and case studies”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 3, pp. 397–415, 2008. DOI: 10.1109/TSMCC.2008.919172.

[21] A. Howard, devrishi, P. Culliton and Y. Guo. “Natural language processing with disaster tweets”. (2019), [Online]. Available: https://kaggle.com/competitions/nlp-getting-started (visited on 29 Feb. 2024).

[22] GitHub. “shrnik/Disaster_Pred”, [Online]. Available: https://github.com/shrnik/Disater_Pred (visited on 15 Mar. 2024).

[23] G. Santafé, J. A. Lozano and P. Larrañaga, “Discriminative vs. generative learning of bayesian network classifiers”, in *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Springer, 2007, pp. 453–464. DOI: 10.1007/978-3-540-75256-1_41.

[24] A. Ng and M. Jordan, “On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes”, *Advances in neural information processing systems*, vol. 14, 2001. [Online]. Available: https://dl.acm.org/doi/10.5555/2980539.2980648.

[25] C. Zhai and J. Lafferty, “A study of smoothing methods for language models applied to information retrieval”, *ACM Trans. Inf. Syst.*, vol. 22, no. 2, pp. 179–214, Apr. 2004, ISSN: 1046-8188. DOI: 10.1145/984321.984322. [Online]. Available: https://doi.org/10.1145/984321.984322.

[26] V. Sanh, L. Debut, J. Chaumond and T. Wolf, *Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter*, 2020. DOI: 10.48550/arXiv.1910.01108.

[27] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. DOI: 10.14257/ijgdc.2013.6.5.09. arXiv: 1412.6980 [cs.LG].

[28] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais and I. Ahmad, “Cloud computing pricing models: A survey”, *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013.

[29] H. Bal *et al.*, “A medium-scale distributed system for computer science research: Infrastructure for the long term”, English, *Computer (New York)*, vol. 49, no. 5, pp. 54–63, May 2016, ISSN: 0018-9162. DOI: 10.1109/mc.2016.127.

MLVU Information Sheet

****Important Note**** Our team consisted of 4 members, less than the regular amount. We tried to find a fifth member for the project but when we stated our project's ambition to potential candidates, they were not willing to participate and left to find another group.

Group number 134

Authors

Name	Student Number
Dovydas Vadišius	2744980
Laura I.M. Stampf	2701672
Lennart K.M. Schulz	2734873
Maria P. Jimenez M.	2692270

Software used The simple model used in the project is a Markov model implemented from scratch. See Section 4.1-III for implementation details. Libraries that we used throughout the project include the following:

- `os`: used to create OS-independent paths for file loading and saving
- `pandas`: used to work with the CSV dataset
- `matplotlib`, `seaborn`: used to create graphs and confusion matrices for data visualization
- `collections`: used the `defaultdict` data type to assign a default value to non-existent entries instead of throwing an exception
- `sys`, `time`: used for benchmarking and debugging
- `math`: used for logarithms in formulas for the Markov model
- `multiprocessing`: used to implement the multithreaded version of the Markov model
- `tensorflow`: used to optimize the CPU load when training the DL model
- `keras`, `keras_core`, `keras_nlp`: used for the DL model, contains the DistilBERT model
- `sklearn`: used to split train and validation data

Use of AI tools None

Link to code The project's code can be found at the following link: https://github.com/lkm-schulz/disastAIr.

Group disagreements None