Real or Not? NLP with Disaster Tweets Project

Vegar Andreas Bergum, *vab1g19,* ChangKai Fu, *ckf1n19,* Adam Ghoumrassi, *ag3u19,* PingChun Tsai, *pct1g19.*

***Abstract*—This project paper discusses the use of non-neural and neural methods for binary text classification in Tweets. A range of different feature extractors, using latent semantic analysis, are tested with a wide range of models. Bayesian optimisation is used to optimise hyperparameters. Gradient Boosting with pre-trained embeddings performs best out of the non-neural methods, while a 2-layer LSTM RNN produces the best neural cross-validated result. Additionally, the BERT model is finetuned for classification, scoring better than all previous models tested.**

I. Introduction

REAL or Not? NLP with Disaster Tweets is a Kaggle competition[1] that provides over $7,000$ labelled tweets. This competition is a binary text classification task: are the tweets concerning a real disaster or not? Twitter is more and more being used as a real-time observational tool, according to the competition description. The motivation behind this competition is to investigate the possibilities of using Twitter as a source of monitoring for emergency services. Additionally, this competition serves as a good example of a getting-started with advanced machine learning techniques when working with highly contextual and sequential data.

As the data is highly contextual and sequential, feature extraction becomes a crucial step in being able to accurately classify the data. This project report includes an overview of a few different latent semantic analysis techniques. Including term frequency-inverse document frequency (TF-IDF) [1] bag-of-words models, pre-trained word embeddings [2] and self-trained character-level embeddings. With a sophisticated Bayesian optimisation approach to hyper-parameter tuning, we train and evaluate a large number of linear, non-linear and neural classifiers. Using the different feature extraction methods we compare the different classification techniques. Additionally, we investigate the use of large-scale pre-trained language models such as BERT [3] to classify the tweets.

II. Feature extraction & latent semantic analysis

Before conducting any sort of machine learning on text-based data, we must first convert it into a structured numerical form that can be processed by a computer. Latent semantic analysis [4] is a technique used in natural language processing to analyse the semantic similarity between textual documents. This requires a latent space representation of tokens in the text, often at a word-level, that captures likeness in meaning. With such a representation, simple techniques like cosine angles between token-vectors can reveal similarity in semantics and contextual appearance.

Traditionally, latent space representations were done at a document level where each dimension represents a semantic token in the document. TF-IDF is one such representation that vectorizes a document into a v-dimensional space, where v is the size of the semantic vocabulary. The vector representation is calculated by equation 1 for each token t in a document d

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t) \tag{1}$$

where the inner functions are defined as $\text{idf}(t) = \log\left[\frac{n}{\text{df}(t)}\right] + 1$ and $\text{tf}(t,d) = \log[1 + f_{t,d}]$ where $f_{t,d}$ is the number of occurrences of token t in document d. With this model, we have one dimension for each unique word in the vocabulary.

Other neural-based methods known as *word embeddings* are widely used in many text applications or natural language processing models. GloVe [2] is one such model, which is trained in a self-supervised manner, encoding vectors based on the contextual placement of tokens in a text. This results in a vector representation where tokens that appear in similar contexts have a small cosine angle between them.

A. Character Embeddings

Tweets often incorporate highly irregular language, resulting in many words in our corpus not having a corresponding GloVe embedding. This decreases the amount of information we have available for use in the model. In an attempt to combat this, we investigated the use of character vectors instead of word vectors. As there are only a small handful of available characters, this makes it possible to have an embedding for every possible character, therefore no information is lost.

We constructed the character vectors by training a word2vec model on the Sentiment140 dataset [5], a corpus of 1.6 million tweets, treating each individual character in a tweet as a separate token. Unfortunately, while this method was able to perform better than random guessing, the performance of this embedding type was fairly weak across all tested models, with the highest performing character embedding model having an accuracy of 66%.

However, this does suggest that the character embeddings do store some useful information about the classification of the tweet, therefore, in future works, it may be beneficial to train a neural network approach that uses a combination of word and character embeddings as inputs.

In the models presented we use TF-IDF features, pre-trained GloVe word embeddings, self-trained character embeddings and BERT's built-in tokenizer.

III. Bayesian Optimisation of Hyperparameters

Many of the machine learning algorithms that we considered as part of this study have hyperparameters that can be adjusted to change their functionality. Constructing the optimal model for our problem requires that many combinations of these parameters be iterated over to find the

[1] https://www.kaggle.com/c/nlp-getting-started/overview

parameters that allow the models to generalise best. This problem requires the use of a parameter searching strategy.

When considering how to conduct parameter search, two considerations must be taken:

Firstly, the search strategy should investigate a reasonable enough portion of the parameter space that we can be confident that we are not omitting a combination of parameters with a significantly improved performance.

Secondly, the strategy should not require excessive computation to find the optimal combination. At each iteration of the search strategy, a machine learning model will be run on our entire dataset, which uses a significant amount of both time and resources to implement. Due to our limited computing resources and pressing deadline, it would be impossible for us to perform an exhaustive search of the parameter space for each of our models.

Given these considerations, we chose Bayesian optimisation as our search strategy. This technique involves computing a probabilistic model that estimates the score of each combination of parameters in the space and iteratively updating this model by computing the output of a set of parameters if the uncertainty of this outcome is high [6]. Over time the optimisation model moves from "exploration" where it computes the model with parameters that it is unsure of the outcome for, to "optimisation" where it computes the model with parameters that it believes will maximise the outcome.

We ran Bayesian optimisation on each of our models using the Python package `scikit-optimise` [7], optimisation was conducted using 10% of the data for testing and separating the remaining 90% in a 80:20 split using 5-fold cross-validation and using testing accuracy as the evaluation metric. The process was initialised to run for 50 iterations, meaning that each model would be run 250 times. As this would still take a considerable amount of time, we decided that for neural networks we would only use cross-validation on the best set of parameters dictated by the optimisation strategy, thereby requiring neural network approaches to only be run a maximum of 50 times. Also, we implemented an early stopping strategy, whereby if the best 3 accuracy scores of the optimisation strategy for a model were within 0.1% of each other, the optimisation would be aborted as it would appear likely that this score is a ceiling past which models are unable to improve.

IV. Data Augmentation

The dataset supplied for this problem consists of a fairly small sample of observations, therefore, we decided that it would be useful to find a way to enrich our dataset by adding some altered samples.

To do so, we made 5 copies of each observation in the dataset, with each word in the copy having a random chance to change to a related word. The related word was chosen at random from the 5 closest words to the source word in the GloVe embedding space. This quintupled the size of our dataset. It should be noted that we were careful to assure that the data augmentation step was only used on the training set and not the full dataset as this would result in many observations being in both the training and testing sets, thereby biasing results. This method was tested on our CNN model and gave a minor accuracy gain of 0.5%. However, due to the computational complexity of our method and the relative unimportance of this addition, it was decided to omit augmentation from our official results.

V. Non-neural methods

To create a baseline for comparison we built a simple cosine angle similarity model based on a TF-IDF representation as described in the previous sections. The training phase consisted of building two TF-IDF corpus matrices, one for each class. In predicting the class of a new document (tweet), it was also transformed into a TF-IDF vector. Te corpus matrix with the lowest cosine angle to the new document determined the predicted class of the document. This produced a baseline scoring accuracy of exactly 70%, using 5-fold cross-validation. Table I shows a small selection of the best performing methods with a handful of feature extractor and parameter combinations.

With this baseline, we started implementing several relevant non-neural classifiers. Firstly, a probabilistic classifier in the form of a Naive Bayes classifier. The Naive Bayes classifier uses Bayes' theorem with the independence assumption to separate the features. Even though Naive Bayes is traditionally known to work well in comparison to SVMs and other similar models with token frequency-based features, its prediction accuracies are quite low. Using both TF-IDF and the character embedding produces accuracies under 60%. However, with the pre-trained 200-dimensional GloVe embeddings it beats the baseline with 1.5 percentage points.

To represent ensemble methods, a Gradient Boosting classifier and Random Forest classifier was trained. Table I illustrates how both methods outperform the baseline and Naive Bayes by over 8 percentage points with the GloVe embeddings. Additionally, both outperform Naive Bayes quite drastically using the character embeddings and TF-IDF vectors. Interestingly, even though both the Random Forest and Gradient Boosting methods are based on decision trees, gradient boosting performs slightly better with a lower max-depth of the underlying trees.

The dataset only consists of $7,613$ samples. The Support Vector Machine (SVM) is known to perform well on small datasets and provides both linear and non-linear classification methods by the use of kernels. We train it using only frequency-based features (TF-IDF) and investigate three different kernels: linear, radial basis functions and polynomial. With a maximum accuracy score of 74.4% with the linear kernel its clearly the best performing model with the TF-IDF vectors.

VI. Neural methods

Utilizing deep neural methods that are capable of exploiting the sequential nature of the data will be crucial to improve upon the non-neural methods we have discussed so far. The two most commonly used architectures for this are recurrent neural networks (RNN) and convolutional neural networks (CNN).

Model	Acc	σ Acc	LSA	Depth	γ	Kernel	Optim	Hidden	#layers	lr	Momemtum	Dropout	Filters	#filters
CosineAngle	**0.700**	0.016	TFIDF											
	0.573	0.025	CHAR											
NaiveBayes	**0.715**	0.025	GloVe											
	0.591	0.015	TFIDF											
	0.692	0.036	TFIDF	8.0	0.0									
GradientBoost	0.643	0.009	CHAR	5.0	0.373									
	0.789	0.014	GloVe	3.0	0.032									
	0.666	0.015	CHAR	8.0										
RandomForest	0.583	0.04	TFIDF	6.0										
	0.786	0.011	GloVe	10.0										
	0.744	0.021	TFIDF		0.003	Linear								
SVM	0.738	0.012	TFIDF		0.894	RBF								
	0.681	0.029	TFIDF		0.077	Poly								
	0.817		GloVe				Adam	100	2.0	1e4	0.99	0.268		
LSTM	0.654		CHAR				SGD	150	3.0	1e3	0.5	0.47		
	0.814		GloVe				SGD	100	2.0	1e4	0.99	0.273		
LSTM+CV	**0.801**	0.015	GloVe				Adam	100	2.0	1e4	0.5	0.267		
	0.688		CHAR				Adam			1e3	0.5	0.351	(6,6,6)	107.0
CNN	**<u>0.845</u>**		GloVe				Adam			1e4	0.99	0.5	(4,4,4)	150.0
	0.799		GloVe				SGD			1e4	0.9	0.207	(3,3,3)	168.0
CNN+CV	**0.799**	0.014	GloVe				Adam			1e5	0.5	0.119	(5,4,3)	177.0
	0.828		BERT				Adam	768	12.0	5e5		0.1		
BERT+CV	0.804		BERT				Adam	768	12.0	2e5		0.1		
	0.813		BERT				Adam	768	12.0	5e5		0.1		

TABLE I: Top Bayesian Optimisation Results

The best performing LSTM scores 81.7% accuracy using the GloVe embeddings and the Adam optimiser with a learning rate of $1.0e04$ and 0.99 momentum. This is a 2-layer bidirectional LSTM network with a hidden dimension of 100. The model was only trained for 6 epochs, as the validation loss started rapidly increasing at the same time as the accuracy decreased. Even with a 26.8% dropout layer, it was clear that the model is very sensitive to overfitting. When training with 5-fold cross-validation, we get an average result of 80.1% accuracy, denoted LSTM+CV in table I. This is the overall best cross-validated result we were able to produce from the Bayesian optimisation routine.

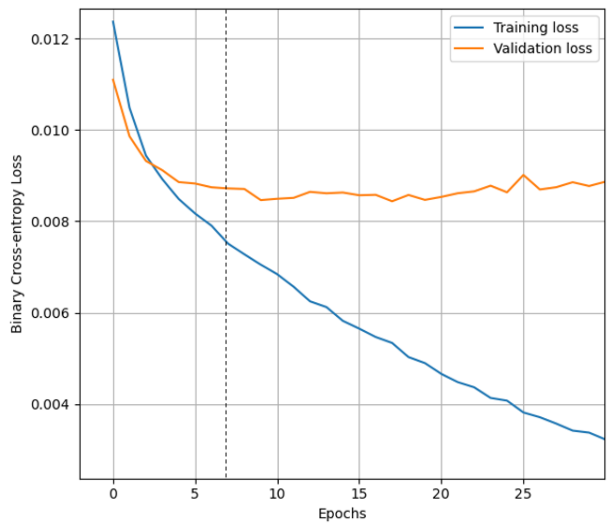

Fig. 1: CNN loss during training. Training loss (blue) and validation loss (orange). Dashed vertical line indicating point of early stopping for the best performing model.

In addition to training the RNN, which are traditionally the best performing models on sequential data, we train a convolutional neural network. CNNs are usually used for image-based data due to its translation invariance and ability to capture spatial features. Using a kernel width that matches the embedding dimensions used, the convolutions become a sliding window across the sentence. This does, in essence, create a one-dimensional convolution over the tokens in the sentence, because the width matches the full word representation. Similarly to the RNN, figure 1 illustrates the history of the loss during training. We notice that the validation loss grows rapidly after just a few epochs. Figure 2 illustrates this further where we can observe that the validation accuracy decreases after just a few epochs. For brevity only the training history has been shown for the CNN - the RNN's training history is remarkably similar.

The CNN ends up with a result that is within the margin of error of the LSTM's cross-validated results, at 79.9% accuracy (denoted CNN+CV in table I). However, a lucky non-validated training run of the CNN was able to achieve 84.5% test accuracy.

VII. Transfer learning

Unlike the neural methods where the model is trained from scratch, transfer learning utilised a pre-training - fine-tuning process to gain better performance with less data. Google's BERT (Bidirectional Encoder Representations from Transformers) is probably one of the most representative pre-trained models. Firstly, BERT is trained bi-directionally to catch the embedding of each word in the sentence by feeding countless online text corpus such as Wikipedia to complete two tasks, i.e. Masked Language Model (MLM) and Next Sentence Prediction [3] to gain maximum 334 million parameters. In the second phase, the model is fine-tuned through the dataset owned by the users to implement downstream tasks, such as document classification, dependency parsing, question answering and name-entity recognition.

Since this is a binary classification NLP task, "bert-base-uncased" pre-trained model is chosen as it is built to

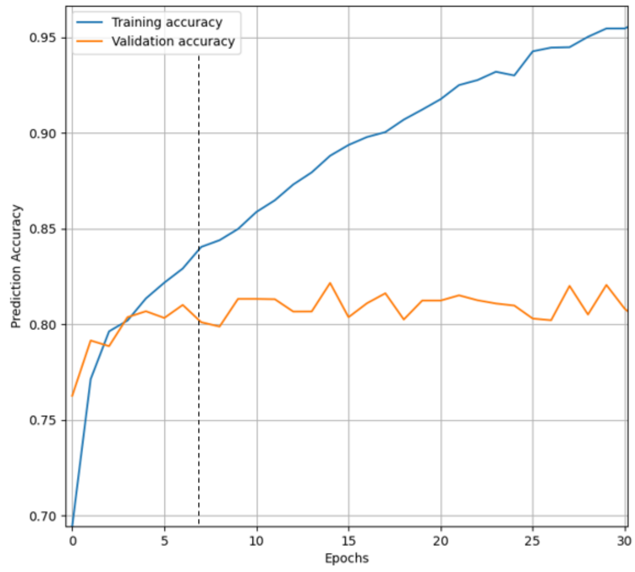

Fig. 2: CNN accuracy during training. Training accuracy (blue) and validation accuracy (orange). Dashed vertical line indicating point of early stopping for the best performing model.

process lower-cased English text. Its architecture has 12-layer transformers, 768 hidden and 12-heads, which means it has 110 million parameters. From figure 3 we could conclude that epoch 3 should be the result parameter as accuracy sharply increases at epoch 4. The best result gained is 82.8% accuracy using the Adam optimiser with a learning rate of $5e-5$ after 3 epochs, which is within our expectation in the scope of transfer learning.

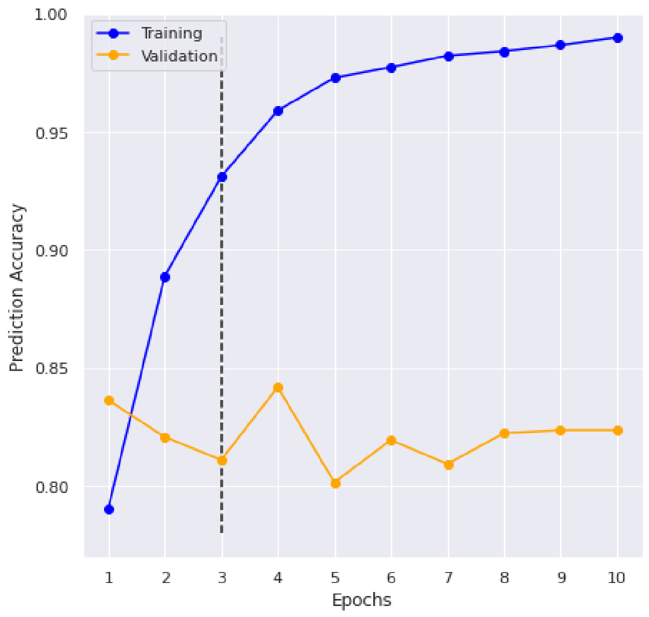

Fig. 3: BERT accuracy during training. Training accuracy (blue) and validation accuracy (orange). The best performing model is achieved given the value of epoch is three.

VIII. Discussion

For the performance of feature extractors, it is obvious that the models processed by the GloVe embeddings achieve better performance than both TFIDF and character embeddings in all possible observations. In comparison, TFIDF is more reliable than character embedding in NaiveBayes and GradientBoost models, but the situation is opposite in RandomForest. This is probably due to the nature of RandomForest which makes a more precise decision with a larger amount of crucial information instead of using the frequency and inverse-frequency to represent each word.

As the training labels for the Kaggle competition were leaked, comparing our results to the leaderboard is useless. The top 50 submissions all have a perfect score of 100% accuracy. However, by manually inspecting some of the published submissions it is clear that our best cross-validated result of 82.8% test accuracy is comparable. By using cross-validation and a generous validation and test-set split we can ensure that the results are representative as the models' ability are generalised. The best CNN performance at 84.5% could be considered an outlier as this was not reproducible in a cross-validated setting.

IX. Conclusion

Within our work, three feature extractors, eight models and various hyperparameters generate over 100 combinations. It is not surprising that BERT (82.8% validation acc.) perform the best within these models. As expected the LSTM model catches up tightly right after at 80.1%. In general, the neural methods perform better than the non-neural ones with a 80+% threshold. It is also predictable that neural methods gain better results with reasonable deeper architectures. Although transfer learning gives a relatively easy milestone without any required pre-processing, it is still promising to investigate neural models further to see if a more reliable architecture could be created to perfectly solve critical problems such as disease classification.

References

[1] G. Salton and M. J. McGill, "Introduction to modern information retrieval," 1986.
[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162
[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.
[4] S. T. Dumais, "Latent semantic analysis," *Annual Review of Information Science and Technology*, vol. 38, no. 1, pp. 188–230, 2004. [Online]. Available: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440380105
[5] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N project report, Stanford*, vol. 1, no. 12, p. 2009, 2009.
[6] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.
[7] G. L. I. S. f. Z. V. . F. Tim Head, MechCoder, "scikit-optimize/scikit-optimize: v0.5.2 (version v0.5.2)," 2018.