Using a Large Language Model to Improve the Performance of Simpler NLP Classifier Models – Michael Szczepaniak

April 20, 2024

1. Executive summary

This project investigated whether a large language model (LLM) could be used to improve the performance of simpler text classifiers by augmenting the data used to train those classifiers. The publicly available ChatGPT 3.5 turbo was chosen as the LLM to augment the publicly available kaggle Disaster Tweets dataset. This approach was tested by building logistic regression and single-hidden-layer neural network models and assessing their performance after being trained on two sets of data: 1) the original data set and 2) the original data set augmented by the LLM which was provided a prompt designed for this task. Both models built with the augmented data added to the original training data outperformed the same model trained on just the original training data by a small amount.

2. Introduction

2.1 The problem of not enough data

This project sought to answer the research question:

Can a pre-trained transformer-based model (i.e. a large language model or LLM) be used to improve the performance of simpler non-transformer-based (NTB) classifier models (e.g. logistic regression or a neural network classifier) by augmenting its original training data?

To answer this question, the following hypothesis was tested:

(performance of models trained with augmented training data) > (performance of models trained with non-augmented training data)

2.2 How the hypothesis was tested

The following 4 scenarios were constructed and compared

- Logistic regression model trained on the original training data only.
- Logistic regression model trained on the original training data plus augmented data. The augmented data had the same number of samples as the original data, so this effectively doubled the size the training data in this scenario.
- Neural network model trained on the original training data only.
- Neural network model trained on the original training data plus augmented data effectively doubling the size the training data in this scenario.

A simple neural network with a single fully connected hidden-layer was chosen. The number of hidden-layer nodes (100) and the hidden-layer activation function (ReLU) were determined by 10-fold cross-validation (see Appendices L and M).

2.3 Definitions and Metrics

original (non-augmented) - This is the base case where only the originally provided data is used to train the simpler NTB classifier models.

augmented - In these scenarios, the original training data is doubled in size by prompting an LLM to provide a sample that is similar to each of the original training samples regardless of class. The LLM was free to decide what “similar” means.

metrics - The area under the Receiver Operating Curve (ROC AUC) on a portion held out from 20% of the original training data and the overall accuracy on the unlabeled test data provided by kaggle (see section 2.6 and Appendix B) will be the metrics used to evaluate the hypothesis against two models (logistic regression and single hidden layer neural network).

2.4 Data for Natural Language Processing (NLP)

The field of Natural Language Processing (NLP) and the related sub-field of Computation Linguistics arose from the need to provide computers with the ability to process and manipulate human language. The kinds of tasks which NLP has enabled range from Named Entity Recognition (NER), Part of speech (PoS) tagging, sentiment analysis and machine translation to text generation such as those used by automated chatbots built from large language models (LLMs). Descriptions of these tasks can be found in Appendix A.

Each of these tasks require the construction of some form of language model. For these models to be useful, they must first be trained to learn patterns in language that are relevant to the task they were designed for. This training requires large amounts of data. Typically, the more data that is available to train a model, the more reliable and accurate that model will be.

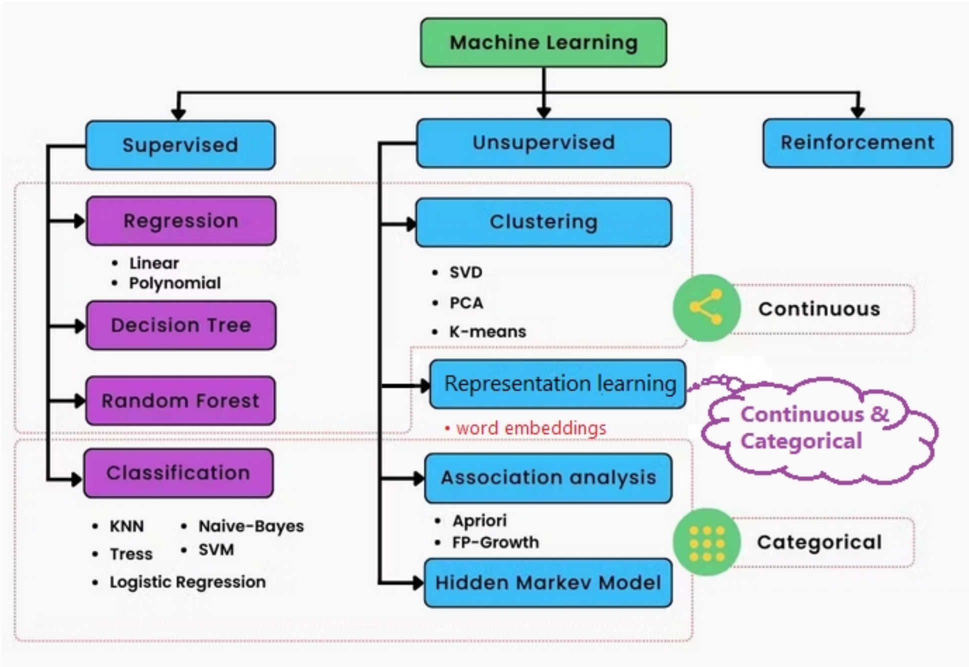


Figure 1. Overview of machine learning tasks

2.5 Data requirements for supervised tasks

Machine learning tasks such as those described earlier, generally falls under three categories: supervised, unsupervised and reinforcement learning as shown in Figure 1. adapted from [1]. Unsupervised learning is typically associated with tasks such as clustering which includes dimensionality reduction for continuous variables, association analysis for categorical variables and representation learning [2] for either continuous or categorical variables. Unsupervised learning does not require labeled data. Reinforcement learning can be considered a semi-supervised task which requires data to construct a useful policy.

Text classification was selected for investigation in this project because it is among the most commonly performed tasks in NLP [3]. These types of tasks range from sentiment analysis and determining relevance of text to determining if a tweet is related to a disaster in need of a response or not. Text classification falls under supervised learning which means that it requires a sufficient amount of labeled data to build effective models. However, this labeled data is typically in short supply because it is typically created from a human-driven curation process.

2.6 Data sources

Until recently, it required many man-hours to generate enough labeled data to effectively train a text classification model. Because this manual process is both slow and expensive, it motivated several automated techniques for generating labeled data which are described in Appendix O. The limitations that came with these techniques provided the motivation for this project.

Two types of data are used in this project which are referred to as *base data* and *augmented data*. The base data is obtained from the ongoing kaggle competition titled "*Natural Language Processing with Disaster Tweets*". Details about this data are described in Appendix B. The augmented data was generated from the large language model (LLM) ChatGPT 3.5 turbo as described in section 5.

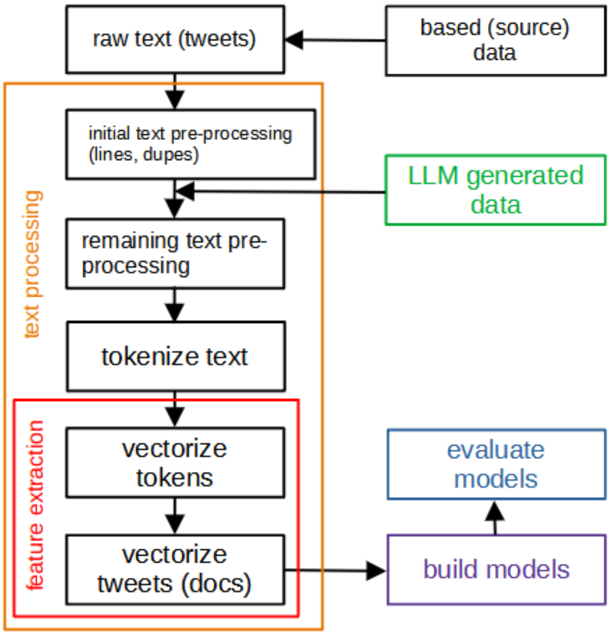


Figure 2. General Project Workflow

3. Text Pre-Processing

3.1 Pre-tokenize steps

Before any model using text as input can be trained, each sample of text must be converted to a numeric representation otherwise known as a vector. However, before this vectorization is performed, text must be pre-processed to a form which facilitates this conversion.

The first step in this process is ensuring that each sample is on its own row and that duplicates are removed. This is the first processing step after extracting the text data related to each sample (tweets) as shown in Figure 2. This step was not necessary for the LLM generated tweets because these tweets were captured in code and could easily be formatted to a single line.

Two kinds of duplicates were discovered in the base data that needed to be handled. These duplicates arose from the choice to only use only the content of the tweets (**text** column) as input features to our models and to ignore the other two prospective features (**keyword** and **location** columns). The two types are referred to as *text-target duplicates* and *cross-target duplicates*. Details regarding the description of these two types of duplicates and how they were handled are described in Appendix C. Neither of these types of duplicates were an issue for the augmented data which is why they didn't need to be handled as shown in Figure 2.

The remaining pre-processing steps are described as follows:

1. URLs were replaced with a special `<url>` token. This token was chosen because it had a mapping to a vector in the embeddings file used for vectorization.
2. Twitter-specific characters such as `@` and `#` were replaced with the special tokens `<user>` and `<hashtag>` respectively. These tokens also had mappings in the embeddings file.
3. Contractions were expanded. For example, "I'm" was expanded to "I am".
4. Digits were replaced with the `<number>` token, stop words were removed (see Appendix D for details) and punctuation was removed.
5. The remaining text was then lemmatized (see Appendix E for details) and the resulting text was converted to lower case.

Each of the steps just described were performed by a python function. A description of all the functions used in the project and what they do are described in Appendix F.

3.2 Tokenize text step

After performing all the steps described in section 3.1, a small number of tweets turned into empty strings: 7 in the training data, 2 in the augmented data and 3 in the test data. These tweets required the addition of the empty string token to the vocabulary as described in Appendix G. Once this was done, these tweets were manually replaced with the empty string token in these pre-processed tweets.

The remaining non-empty tweets were checked for word counts and whether a mapping to an embedding existed. Words that occurred only once (aka *singletons*) in the base (original) training data were dropped because they were assumed to carry little if any useful information. Words that did not have a mapping to an embedding were also dropped because these words could not be vectorized. After applying these two criteria, the vocabulary was roughly cut in half being reduced from over 10000 to 4872 word / tokens.

With the vocabulary established and out-of-vocabulary tokens removed, the remaining text was parsed into lists storing each token as a separate element in preparation for vectorization.

4. Exploratory data analysis (EDA)

Exploratory data analysis (EDA) was performed at two points in the workflow described in Figure 2. The first EDA was done after the ***initial text pre-processing*** step and the second was done after the ***tokenize text*** step. The first EDA looked at class balance and tweet length distributions by class. The second EDA looked at most frequently occurring tokens in each class.

4.1 Class balanced

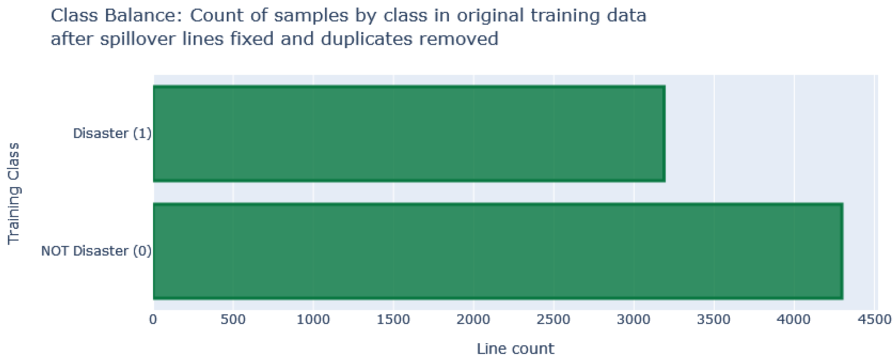


Figure 3. Tweet counts in each class of the original training data

While overall accuracy is the most intuitive metric to compare models, it can be misleading. This is because two models with similar accuracies may assign probabilities to the correct and incorrect classes very differently. In such a case, the model that assigns higher probabilities to the correct class and lower probabilities to the incorrect class on average would be the superior model. When classes are relatively balanced, the area under the ROC curve (ROC AUC) is a good metric to use to compare classifiers. This is because this metric will be higher for a model that assigns probabilities more accurately than the model which assigns probabilities less accurately.

As shown in Figure 3., the two classes are relatively balanced. The NOT Disaster tweets (57.4%) outnumber the Disaster tweets (42.6%), but the difference is only about 15%. This check provided good evidence that using ROC AUC as a metric to compare models was a reasonable choice.

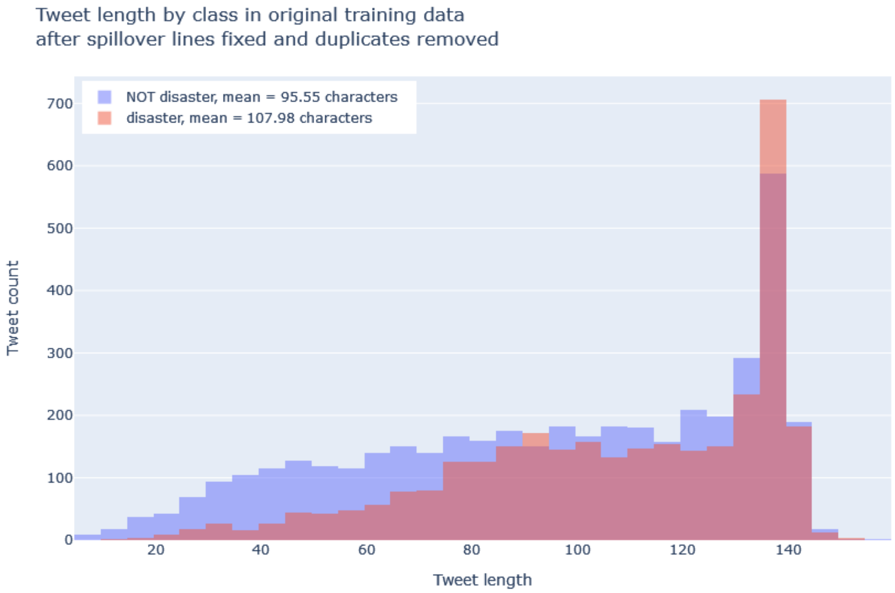


Figure 4. Tweet character length by class

4.3 Tweet Length distributions

The length of disaster-related tweets tended to be longer than the NOT disaster-related tweets as shown in Figure 4. This implies that tweet length would likely be a reasonable feature to use in the context of modeling where the goal is predictive performance.

4.4 Most frequently occurring tokens by class

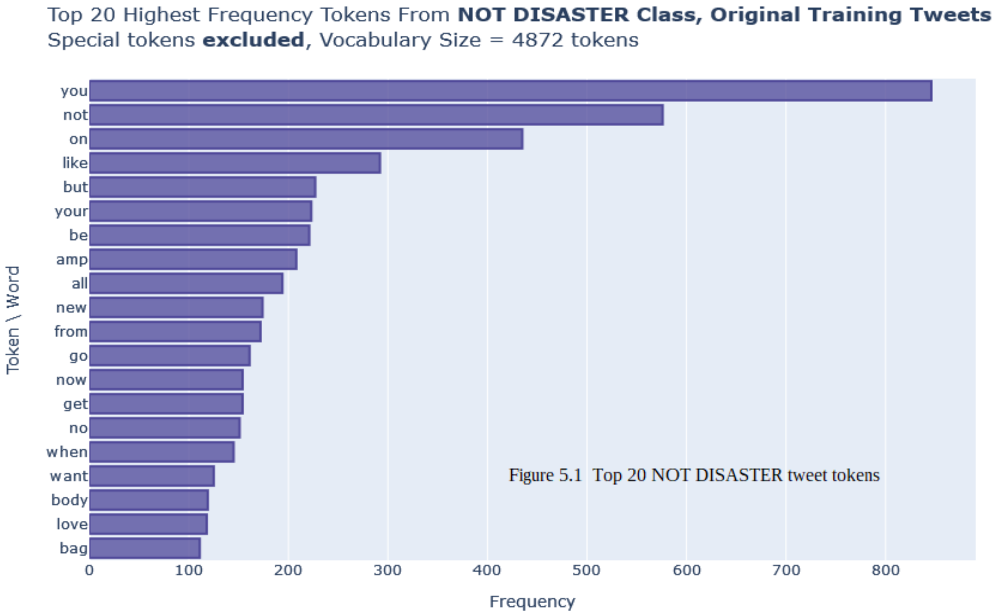


Figure 5.1 Top 20 NOT DISASTER tweet tokens

As shown in Appendix H, the 4 special tokens described in section 3.1: <url>, <hashtag>, <user> and <number> were the top 4 tokens most frequent in each class. When the special tokens are removed, we see the distributions shown in Figures 5.1 and 5.2

The result of removing words like “you”, “not” and “on” from the stop word list shows up in both of these barplots. In hindsight, it probably would have been better to leave them in the list.

The non-stop words are very neutral with respect to the context of a disaster in 5.1 (e.g. amp, new, body, love, etc.) while the non-stop words in 5.2 appear to be much more related to a disaster (e.g. fire, kill, news, disaster, crash, etc.).

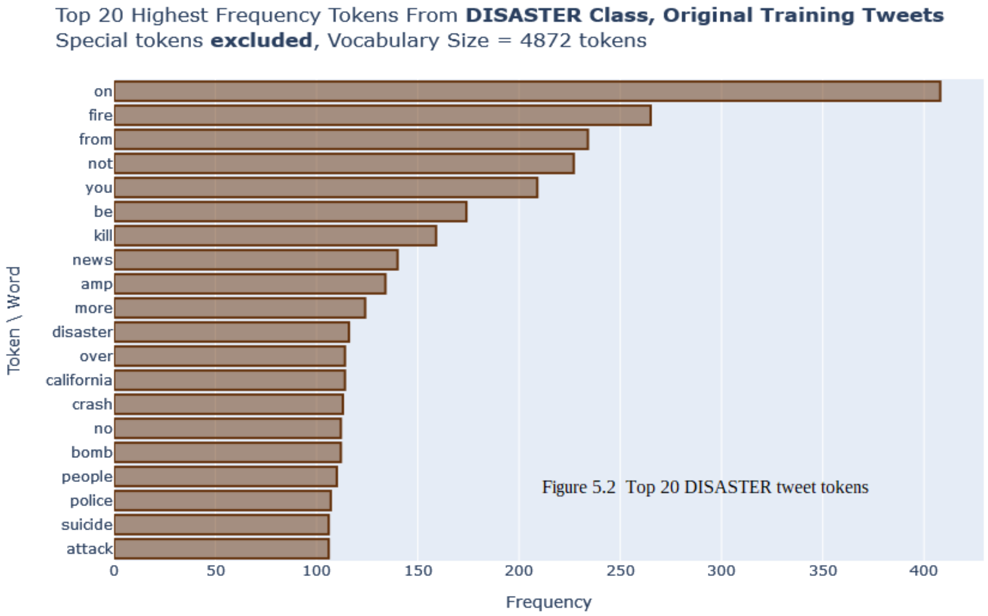


Figure 5.2 Top 20 DISASTER tweet tokens

5. Augmented Tweets Generation

5.1 Approaches to data augmentation

The two main problems which motivate the augmentation of an existing data set are to address a serious class imbalance or to address an insufficient amount of training data from which the model can learn. For structured continuous data, techniques such as SMOTE [4] are available to address the class imbalance problem. For labeled image data, a number of techniques such as cropping, flipping, zooming, rotating and injecting noise into existing labeled images can be used to address both the problems of insufficient training data or class imbalance [5].

In the domain of NLP-related tasks, many data augmentation techniques have been developed [6]. Three techniques are described in Appendix O: Back Translation (BT), Easy Data Augmentation (EDA) and using a transformer. BT generates a new word or phrase by first translating text from an original language (e.g. English) into a foreign language (e.g. French) and then translating the foreign translation back to the original language [7] as shown in Figure O1 in Appendix O. EDA is a set of simple and intuitive transformations which can be applied to the original text data to generate augmented samples [8].

As seen in examples listed in Appendix I, LLM tranformers create the kinds of variations that result from the augmentation techniques just described. In addition to those simple variations however, much more sophisticated and realistic variation are generated due to its ability to leverage structural language knowledge it had learned through extensive training. This is why the LLM was chosen to generate the augmented tweets.

5.2 The LLM and its prompt

Augmented tweets were generated from prompts to the ChatGPT 3.5 Turbo LLM. As described in Appendix I, there were 5 iterations on the prompts before the final prompt used to double the training data was implemented. The LLM requires a context and a prompt. The prompt in this project has two components: the instruction prompt and the example tweet. Each class had its own set of (context, instruction prompt). The final versions are listed below. Below, the context is shown in blue text, the instruction prompt is shown in green text and the reference to the example tweet is shown in red text:

NOT DISASTER Context: *You are social commentator who enjoys diverse opinions and likes to tweet.*

NOT DISASTER Prompt: *Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes, but refers to a different activity, feeling and location:*
*<base **non-disaster** tweet appended>*

DISASTER Context: *You are a world class reporter who has observed a disaster and likes to tweet.*

DISASTER Prompt: *Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes, but refers to a different disaster and location:*
*<base **disaster** tweet appended>*

5.3 Expanding the training data

The original training data processed with the line fixes and the duplicates removed was the starting point for the data augmentation. This starting point was used because the pre-processing steps that came after these fixes needed to be applied to both the base and augmented data as shown in Figure 2.

Before prompting the LLM, the base training data was separated by class into two dataframes. Complete prompts (context, instruction part of prompt and the appended tweet) were constructed by appending a tweet obtained by iterating through each class dataframe to the context and instruction prompt designed for that particular class.

After the augmented tweets were generated, the base data and augmented tweets follow the rest of the pre-processing path together starting with the 5 steps described in the ***remaining text pre-processing*** box of Figure 2. After completing these steps, all the data became ready for the vectorization step.

6. Text Vectorization

6.1 Selecting the vectorization method

There are three main ways to vectorize text as described in Appendix J: One-hot encoding, TF-IDF and word embeddings. The Twitter GloVe embeddings were selected for this project because they have been shown to perform well under a wide variety of NLP tasks [9] and because they are easily accessible.

6.2 Word versus document vectorization

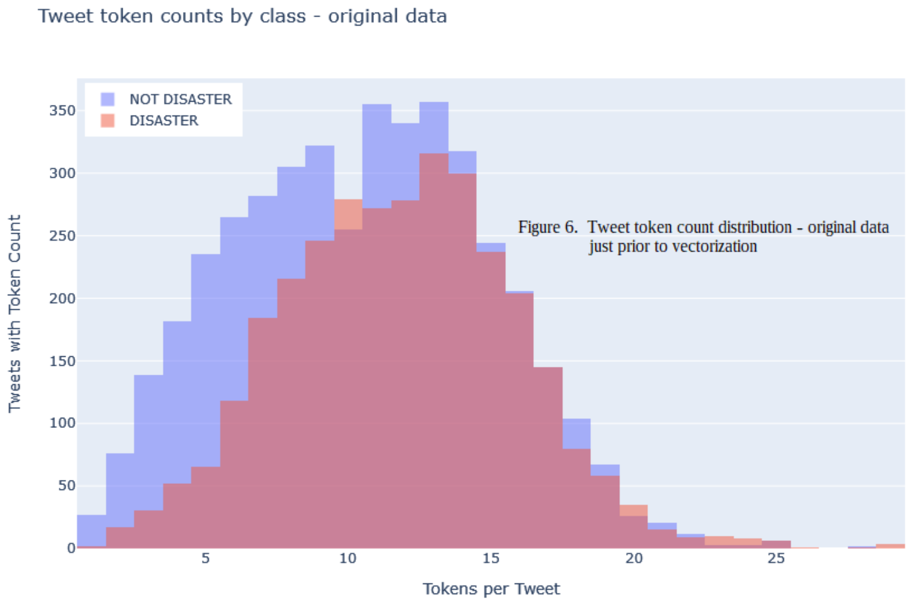


Figure 6. Tweet token count distribution - original data just prior to vectorization

Word embeddings provide vector representations for each token (word) in a text document. A number of options exist to convert a set of word vectors to a representative document vector as described in Appendix K. In the context of this project, each tweet is considered a document and these documents are relatively short due to the 140 character limit that was in place at the time the training data was collected. This limitation translates to a maximum of 29 tokens as shown in Figure 6.

Input vectors used to train our models need to be of fixed length. Selecting the 50-dimension GloVe Twitter embedding and using a 30 token sequentially stacked input resulted in an input vector of 50 x 30 = 1500 dimensions. Through some trial-and-error, this seemed to be a reasonable number of features for the inputs providing both a good amount of capacity (model complexity) as well as ensuring that model training times were reasonable given the available compute resources.

6.3 Input padding

Because each processed tweet contains a different number of tokens, padding was required to account for the unused vector space. This space needed to be filled in a manner that did not add any additional bias to the models. To meet these requirements, the empty space token was selected to fill the input vector dimensions starting from immediately following the last token up through the last position of the input vector which is also described in Appendix K.

7 Logistic Regression Models

Although logistic regression models are generally less accurate than most other types of models, they are often a good starting point in modeling most classification tasks for three reasons. For starters, these are among the most interpretable classifier models available. The coefficients of a logistic regression model tell the user the degree to which the logit of the probability that the sample is in the y=1 class is influenced by a particular predictor. Another advantage of generalized linear models, is that it can be fit to the data directly without needing to determine any hyper or meta parameters.

Finally, in addition to being interpretable and not having any hyper-parameters, logistic regression models establish a reasonable baseline for which more flexible models can be compared. For example, if a logistic regression model is found to be 75% accurate ± 4% and a neural network classifier is determined to be 68% ± 4%, this likely indicates a problem with how the neural network was built and or trained. It is for these reasons logistic regression was chosen as one of the models used for evaluating the research question in this project.

Logistic regression models start with the assumption that the log odds or logit of the conditional probability that a sample is in the target class is a linear function of it predictors. More formally, this can be written as:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \tag{7.1}$$

where *p(X)* = P(y=1 | X) and X are the p predictors. The β coefficients are typically determined by maximizing the likelihood function as described here [10, pages 134-137].

8. Neural Network Models

Neural Networks (NNs) have become a popular choice for many current machine learning tasks in recent years including text classification because their extreme flexibility allows then to learn a well-fitting function from most types of data.

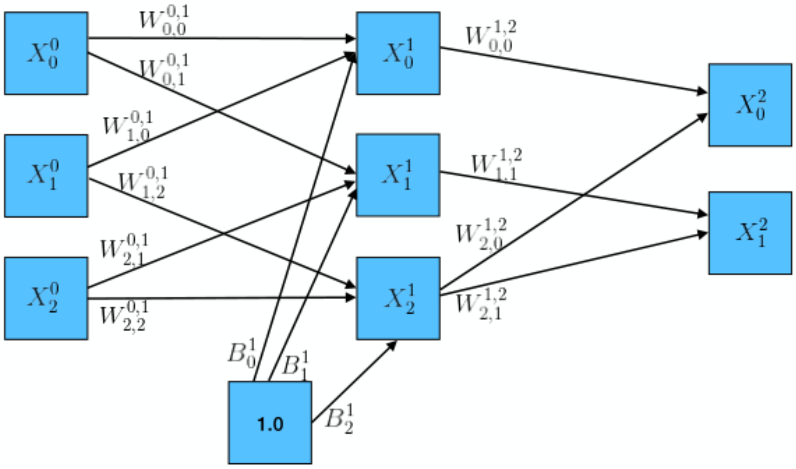


Figure 7. Neural network with single hidden layer Biases (B^i_j) not added to input or output nodes.

From a high-level perspective, they are essentially a class of non-linear functions which map a set of inputs to one of more outputs. They are used in supervised, unsupervised, reinforcement and representational learning problems but most typically for supervised learning.

A simple NN architecture is represented as an acyclic graph with three layers: an input layer, a hidden-layer and an output layer as shown in Figure 7. [11, slide 20]. The outputs of this network are computed by first calculating the values for the hidden nodes and then using them to compute the final outputs. The value of the hidden units are computed by first calculating the sum of each input coming from connections from the previous layer (X^{layer}_{node}) multiplied by a weight associated with the connection ($W^{from\ to\ layer}_{from\ to\ node}$), adding a bias and then applying an activation function g_{layer} ( ) to that sum as shown in Figure 8 [11, slide 32].

$$X_0^1 = g_1((1.0 * B_0^1) + (X_0^0 * W_{0,0}^{0,1}) + (X_1^0 * W_{1,0}^{0,1}))$$
$$X_1^1 = g_1((1.0 * B_1^1) + (X_0^0 * W_{0,1}^{0,1}) + (X_2^0 * W_{2,1}^{0,1}))$$
$$X_2^1 = g_1((1.0 * B_2^1) + (X_1^0 * W_{1,2}^{0,1}) + (X_2^0 * W_{2,2}^{0,1}))$$

$$X_0^2 = g_2((X_0^1 * W_{0,0}^{1,2}) + (X_2^1 * W_{2,0}^{1,2}))$$
$$X_1^2 = g_2((X_0^1 * W_{1,1}^{1,2}) + (X_2^1 * W_{2,1}^{1,2}))$$

Figure 8. Calculating Figure 7. node outputs

8.1 Activation Function

There are many choices for activation functions, each having their own set of pros and cons as described here [11, slides 23-30]. The sigmoid and hyperbolic tangent (tanh) functions were historically the most frequently used, but had been displaced with ReLU and its variants in recent years as networks started getting larger.

Because we are using a single hidden-layer network, we won’t need to worry about the “vanishing gradient” problem [11, slide 14], so a 10-fold cross-validation (CV) was conducted to determine that a ReLU worked slightly better than a sigmoid on our training set. The results of this CV is reported in Appendix L.

8.2 Number of hidden layers and node connections

After selecting the ReLU as the activation function for hidden layer, additional design choices must be made before an NN-based model can be implemented. Other choices include:

i. number of hidden layers
ii. how nodes in each layer are connected
iii. number of nodes in each layer

Because a single hidden-layer NN with a large number of hidden units has the ability to approximate most functions [10, page 407] and state-of-the-art performance on the text classification task was not required, a single hidden-layer architecture was chosen. A fully connected hidden-layer to both the input and output layers was also selected in order to provide the network with an opportunity to learn features related to both the sequence of words as well as relationships between words.

8.3 Nodes in each layer

After deciding on a single hidden-layer that is fully connected, the number of nodes or units in each of these layers needed to be determined. As a general rule, as the number of hidden-layer units increases, the more flexible the NN function becomes and the higher the risk of over-fitting. Conversely, if there are too few hidden units, the NN function may not be flexible enough increasing the risk of under-fitting.

Because there is no way to estimate the proper number of hidden units a priori, a 10-fold CV was done with 100 and 300 hidden units. Because no significant difference was found, 100 was selected for the project as described in Appendix M.

9. Results

Because the kaggle test data is unlabeled, to generate ROC curves, 20% of the of the test data was held out for performance evaluation. In the first of two training scenarios, only the remaining 80% of the training data was used to train a logistic regression (LR) model and a neural network (NN) model. These two models were used to compare results from the second training scenario which used the same held-out 20% of training data for performance evaluation and the remaining 80% of the training data plus all of the data generated from augmentation to train the models. The ROC curves and their respective AUC values are shown in Figures 9. and 10.

9.1 Logistic regression ROC AUC results

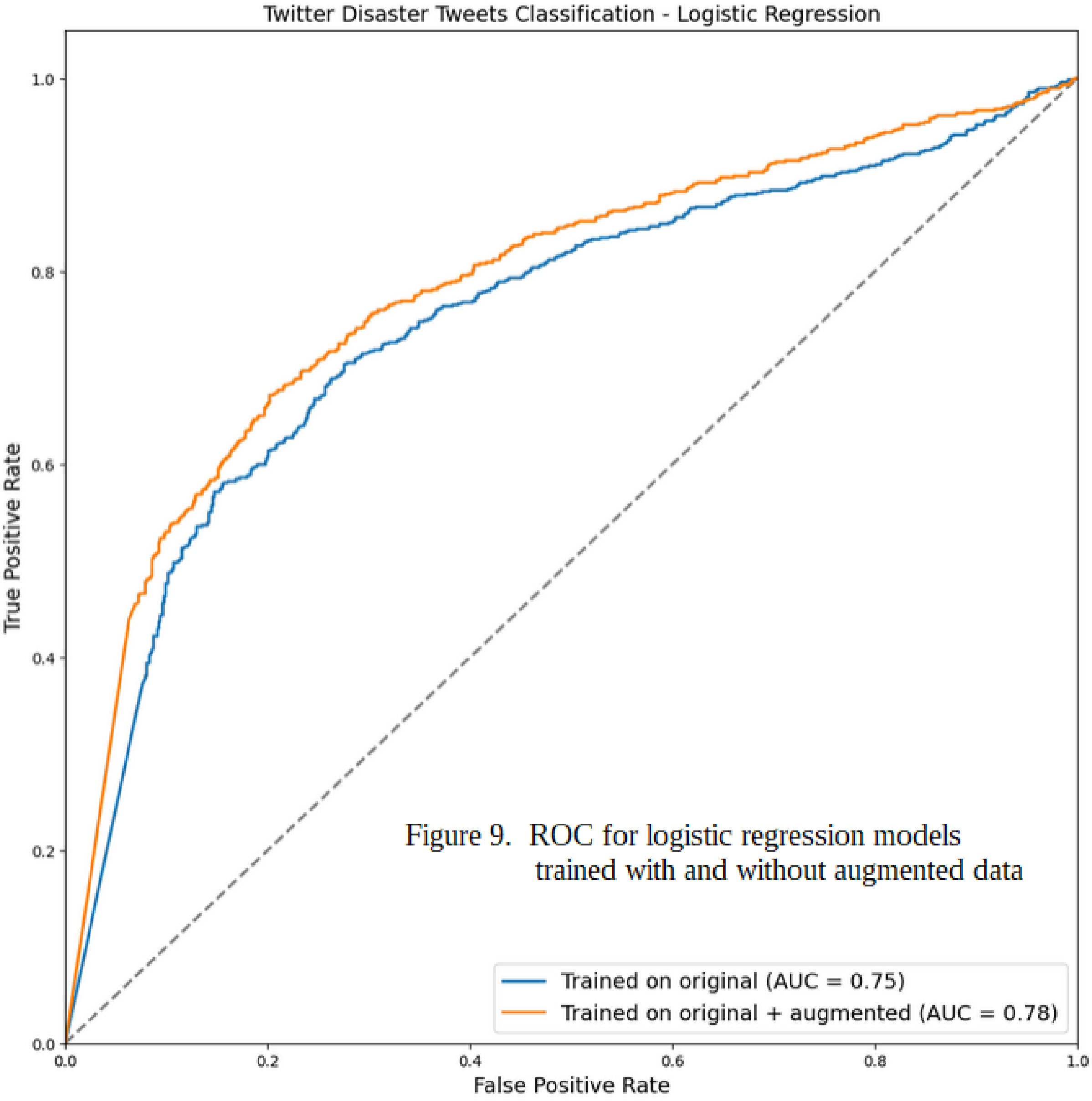


Figure 9. ROC for logistic regression models trained with and without augmented data

Besides having a higher overall AUC, the LR model trained on original + augmented data made better predictions over a vast majority of thresholds as shown by the orange line Figure 9. This indicates that this model clearly does a better job assigning class probabilities than the LR model trained on just the original data (blue line of Figure 9).

9.2 Neural network ROC AUC results

The NN model trained on the original + augmented data also shows a higher AUC as shown in Figure 10, but the difference between this model and the NN trained on just the original data is less pronounced than between the LR models. This smaller difference could be due to a number of factors including lack of tuning or choice of network architecture.

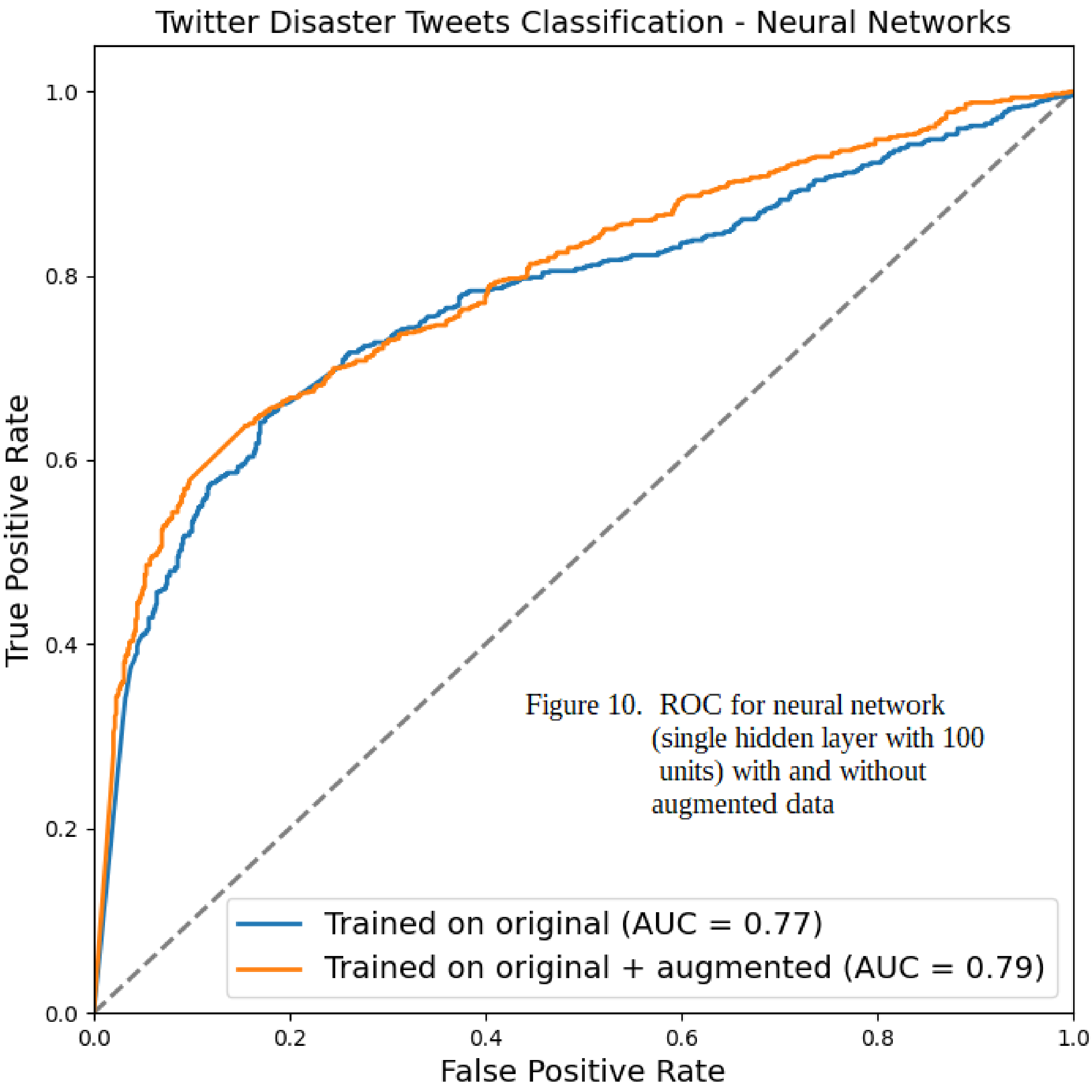


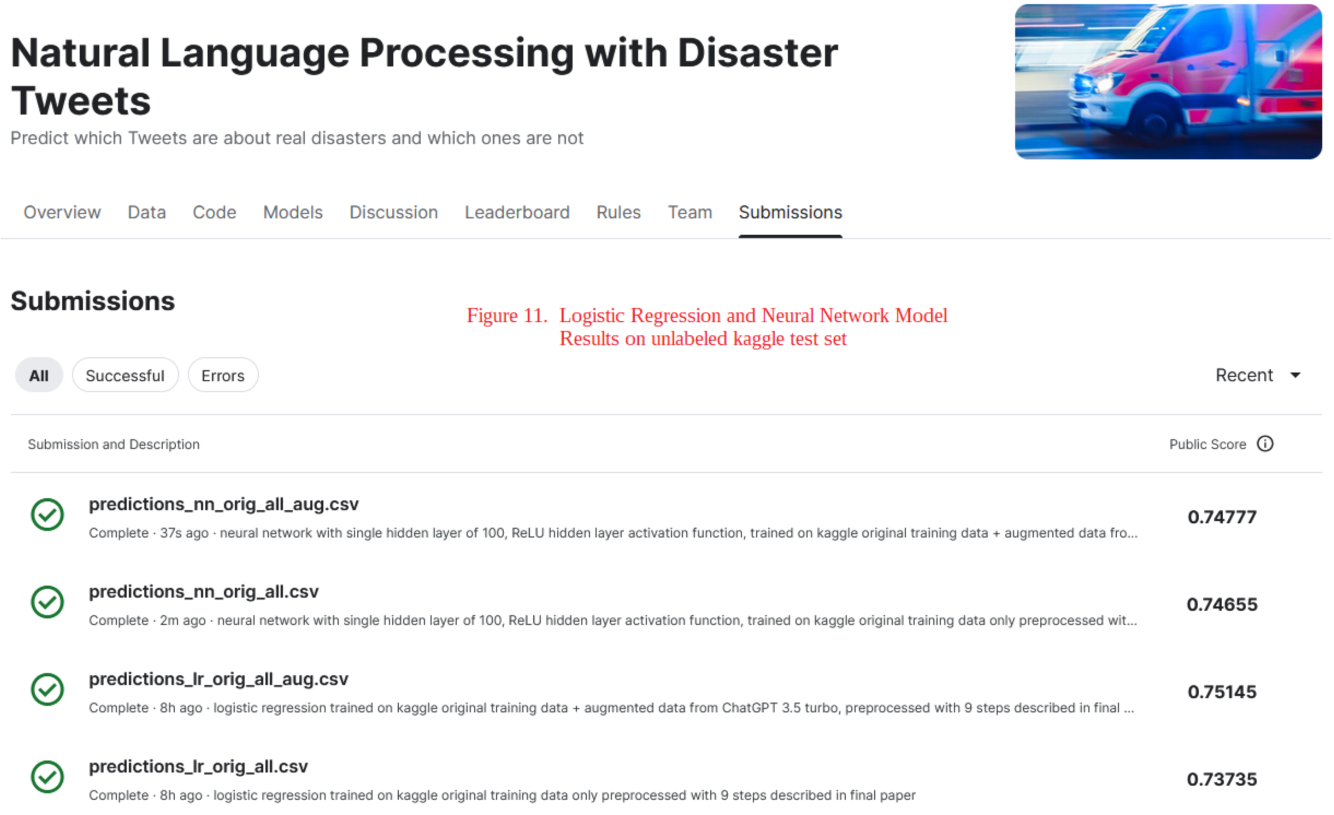
Figure 10. ROC for neural network (single hidden layer with 100 units) with and without augmented data

9.3 Results on the unlabeled kaggle test set

Accuracies for each scenario defined in section 2.2 are shown in Table 1. and Figure 11. below.

Model	Description	Kaggle Score
Logistic Regression (LR)	LR with 1500 features, trained on all original kaggle train data only	0.73735
Logistic Regression (LR)	LR with 1500 features, trained on all original kaggle train data plus augmented data	0.75145
Neural Network (NN)	NN with 1500 features, single hidden-layer with 100 units, ReLU hidden-layer activation function, trained on all original kaggle train data only	0.74655
Neural Network (NN)	NN with 1500 features, single hidden-layer with 100 units, ReLU hidden-layer activation function, trained on all original kaggle train data plus augmented data	0.74777

Table 1. summarizes the results shown in Figure 11.



Natural Language Processing with Disaster Tweets

Predict which Tweets are about real disasters and which ones are not

Submissions

Figure 11. Logistic Regression and Neural Network Model Results on unlabeled kaggle test set

Submission and Description	Public Score
predictions_nn_orig_all_aug.csv Complete · 37s ago · neural network with single hidden layer of 100, ReLU hidden layer activation function, trained on kaggle original training data + augmented data fro...	0.74777
predictions_nn_orig_all.csv Complete · 2m ago · neural network with single hidden layer of 100, ReLU hidden layer activation function, trained on kaggle original training data only preprocessed wit...	0.74655
predictions_lr_orig_all_aug.csv Complete · 8h ago · logistic regression trained on kaggle original training data + augmented data from ChatGPT 3.5 turbo, preprocessed with 9 steps described in final ...	0.75145
predictions_lr_orig_all.csv Complete · 8h ago · logistic regression trained on kaggle original training data only preprocessed with 9 steps described in final paper	0.73735

10. Summary and Conclusion

Models trained with augmented tweets performed slightly better than the same model trained on the original non-augmented corpus. The two highest scores shown in Table 1. and Figure 11. were the models trained on the original data with the augmented data added. The 1.9% difference in accuracy between the logistic regression (LR) models was more pronounced than the 0.15% difference between the neural network (NN) models.

The LR model trained on the augmented dataset outperformed both NN models. While this was somewhat unexpected, it was not a complete surprise given that minimal time was devoted to the design of the network and no tuning was performed.

The results of this project provide evidence in support of the hypothesis posed in the research question stated in section 2.1. Because the differences between models trained on the original data versus models trained on original + augmented data is small, this evidence is not strongly compelling but does suggest that further investigation would likely add to the evidence in support of the hypothesis.

The quality of the training data may have contributed to the lower than expected accuracy of the NN models. As described in Appendix N, a short investigation late in the project indicated that about a quarter of the disaster labeled tweets (class = 1) appear to be mislabeled based on human inspection. This could explain a significant portion of the NN models lackluster performance.

A basic assumption underlying an LR model is that features are assumed to be independent. This implies that the LR model can learn the importance of certain words and even the importance of the position of that word in the tweet, but it can not learn anything about those words in relationship to other words in the tweet. The NN model on the other hand, did have some capacity to learn relationships between words in the tweet by virtue of it fully connected hidden layer.

The combination of the results provided in Appendix N and the fact that the NN did not perform better than the LR model in spite of having two orders of magnitude more parameters, suggests that the training data was not labeled with much if any context in mind. If this was the case, it would well explain the results seen in this project.

11. Future Work

Many interesting questions arose during the course of this project and would be good topics for further inquiry. Eight of these questions are listed below.

- How did the NN models perform on the tweets that were found by human inspection to be mislabeled?

- Would either model perform better if the **location** and **keyword** columns were included as features?

- Would a more sophisticated NN architecture (e.g. a CNN or RNN) show a larger difference when trained on the two training sets (original versus original + augmented)?

- Would different prompts change the character of the augmented tweets in such a way as to improve model predictability?

- How well did the LLM used in this project do on the unlabeled kaggle test data?

- Do the character lengths of augmented tweets distribute similarly to the original training data?

- Would we see a significant difference between models trained on different corpora?

- Would augmenting the data even further improve performance (e.g. tripling or quadrupling the number of training samples)?

- Could a good model be built to accurately identify the mislabeled training tweets? If so, could using this model together with a class prediction model improve the class prediction performance?

- This project assumed that responses from the LLM were always class-appropriate with respect to the prompt it was provided. To what extent is this true? Does the LLM ever provide a response that belongs in the other class? If so, at what rate?

Appendix A – A few important NLP tasks

Named Entity Recognition (NER) – This task can be thought of, on a basic level, as determining whether a word is a "person, place or thing". In practice, additional categories such as "date", "organization", etc. are added to the entity list to be identified by the task.

Part of speech (PoS) tagging – Figuring out whether a word is a noun, verb, adjective or some other part of speech is the objective of this task.

Text classification – This task answers the question: "Does this document, article or some other collection of text belong in category x or category y or ...?. This is the more general term for tasks such as *sentiment analysis* (where x = positive feeling, y = negative feeling) or *spam detection* (where x = spam, y = not spam)

Machine translation – This task translates from one language such as English into another such as Spanish.

Text generation – This is the more general term which encompasses tasks such as *autocompletion*, *text summarization*, *question answering* or applications which dialog with users such as *chatbots*.

Appendix B – The Kaggle Disaster Tweets Dataset

Two types of data were used in the project. The first type is referred to as “base data” which is obtained from the ongoing kaggle competition titled “*Natural Language Processing with Disaster Tweets*” and is currently available at:

https://www.kaggle.com/competitions/nlp-getting-started/data

The base data consists of two files: **train.csv** and **test.csv** which have the following characteristics:

train.csv - This file has 8562 lines of text. Each sample (row) has the following 5 fields (columns):

- **id -** integer, unique identifier for each row which should always have a value
- **keyword -** string, a particular keyword from the tweet which may be blank
- **location -** string, the location the tweet was sent from which may be blank
- **text -** string, the text of the tweet
- **target -** integer, 1 or 0 representing a binary label to be classified and denotes whether a tweet is about a real disaster (1) or not (0)

test.csv - This file has 3700 lines of text. Each test sample (row) has the same first 4 fields (columns) as the train.csv file: **id**, **keyword**, **location** and **text**. There is no **target** column because competition participants are expected to predict and record their predictions of these values as part of their submissions.

Because **test.csv** data are unlabeled, a 20% random sample from the training data was set aside to evaluate ROC AUC to assess model performance. Scoring from kaggle on the unlabeled test data was used to evaluate model accuracy.

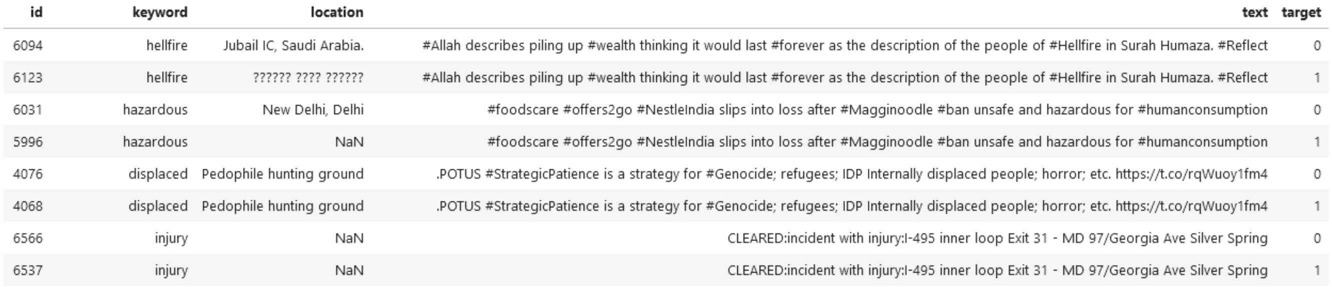
Appendix C – Two types of duplicates

C.1 Text-target duplicates

If two or more rows in the original data had the same values in the **`text`** and **`target`** fields (columns), then these were considered **text-target duplicates**. Because only the content of the tweet (value in the **`text`** field) was used to classify it as **disaster** or **not disaster**, only one of these instances provides useful information for the model to learn. For this reasons, only the first instance of these duplicates was retained and remainder were discarded from the training set.

C.2 Cross-target duplicates

When two rows in the data had the same **`text`** values (tweet content), but different values for **`target`**, these rows were considered **cross-target duplicates**. Examples of these types of duplicates are shown below. Since we did not know which tweet had the correct target value, all of these types of duplicates were removed from the training set.



id	keyword	location	text	target
6094	hellfire	Jubail IC, Saudi Arabia.	#Allah describes piling up #wealth thinking it would last #forever as the description of the people of #Hellfire in Surah Humaza. #Reflect	0
6123	hellfire	?????? ???? ??????	#Allah describes piling up #wealth thinking it would last #forever as the description of the people of #Hellfire in Surah Humaza. #Reflect	1
6031	hazardous	New Delhi, Delhi	#foodscare #offers2go #NestleIndia slips into loss after #Magginoodle #ban unsafe and hazardous for #humanconsumption	0
5996	hazardous	NaN	#foodscare #offers2go #NestleIndia slips into loss after #Magginoodle #ban unsafe and hazardous for #humanconsumption	1
4076	displaced	Pedophile hunting ground	.POTUS #StrategicPatience is a strategy for #Genocide; refugees; IDP Internally displaced people; horror; etc. https://t.co/rqWuoy1fm4	0
4068	displaced	Pedophile hunting ground	.POTUS #StrategicPatience is a strategy for #Genocide; refugees; IDP Internally displaced people; horror; etc. https://t.co/rqWuoy1fm4	1
6566	injury	NaN	CLEARED:incident with injury:I-495 inner loop Exit 31 - MD 97/Georgia Ave Silver Spring	0
6537	injury	NaN	CLEARED:incident with injury:I-495 inner loop Exit 31 - MD 97/Georgia Ave Silver Spring	1

Appendix D – Stop words

The default list of stop words provided by the spaCy package were initially used in pre-processing. However, this list seemed too aggressive for our application, so some of these stop words were removed from the default list before being applied to the tweet text.

No formal process or set of rules were used to determine which stop words would be removed. The list below was derived primarily from intuition.

you	now	last	top	anything	except
on	more	many	most	everything	within
not	over	never	during	sometimes	above
from	some	any	next	serious	below
was	first	everyone	while	everywhere	nobody
but	full	every	call	none	afterwards
your	down	before	very	nothing	anywhere
all	may	under	no	only	when

Appendix E – Lemmatization

Stemming is closely related to **Lemmatization**. Stemming removes or stems the last few characters of a word, often leading to incorrect meanings and spelling. **Lemmatization** considers the context and converts the word to its meaningful base form, which is called a lemma. Sometimes, the same word can have multiple different lemmas. Doing Part of Speech (POS) tagging for the word in that specific context is required for lemmatization. Here are a few examples to illustrate all the differences between the two:

1. If you lemmatize the word '**Caring**', it would return '**Care**'. If you stem, it would return '**Car**' and this is misleading.
2. If you lemmatize the word '**Stripes**' in **verb** context, it would return '**Strip**'. If you lemmatize it in **noun** context, it would return '**Stripe**'. If you just stem it, it would just return '**Strip**'.
3. You would get the same results whether you lemmatize or stem words such as **walking, running, swimming**... to **walk, run, swim** etc.
4. Lemmatization is computationally more expensive than stemming because it involves look-up tables and other processing. Stemming is typically preferred when you have a large dataset and performance is an issue.
5. When accuracy is paramount and datasets are of manageable size, lemmatization is preferred.

Because item 5. was the case with the kaggle Disaster Tweets data, lemmatization was chosen over stemming for this project.

This appendix was adapted from [20].

Appendix F – Python functions used in the project

All the code used for this project can be found in the following git repository:

https://github.com/MichaelSzczepaniak/llmamd

The pre-processing, analyses and model building were all done in python jupyter notebooks. To keep these notebooks manageable, a module was built from which the notebooks could call various functions to perform various tasks. These functions are listed in the **`pytools.py`** file in the top level of the project.

The following are a list of the functions used in the analysis along with a short description of what they do. Further details are available in the docstrings of each of the functions in the module.

FUNCTION	DESCRIPTION
`fix_spillover_lines(list_of_lines)`	Fixes lines read in from a file that should be on a single line
`make_url_counts(list_of_lines)`	Counts the number of urls in each line of list_of_lines and adds that count to the end the line
`replace_urls(list_of_lines)`	Replaces urls in each line of list_of_lines with the text/token <url>. This token was chose because it maps to an embedding vector in the **glove.twitter.27B.50d.txt** file
`expand_contractions(list_of_lines)`	Does a crude expansion of contractions like "I'm" into "I am" for each word in list_of_lines
`replace_twitter_specials(list_of_lines)`	Replaces the @ and # characters with <user> and <hashtag> respectively in the sub-strings within list_of_lines. These replacement tokens were chosen because they map to embedding vectors in the **glove.twitter.27B.50d.txt** file

`spacy_digits_and_stops(`
`df, text_col = 'text',`
`spacy_model="en_core_web_md")`
	Replaces digits with <number> token, removes stop words, removes punctuation, lemmatizes remaining tokens and stores them in lower case

`write_lines_to_csv(list_of_lines,`
`file_name)`
	Writes a list of lines to a file

`get_glove_embeds(embed_path,`
`embed_as_np)`
	Reads in and returns the set of embeddings as a dictionary where the key is a word or token and value is the embedding vector

`get_tweets(tweet_file_path)`	Reads a processed tweets data file and returns a dataframe with just the column with tweet text.
`remove_oov_tokens(vocab, list_of_lines)`	Replaces each out-of-vocabulary (OOV) token with an empty string

FUNCTION (cont.)	DESCRIPTION (cont.)
get_prompt_setup(prompt_date, prompt_log_path)	Gets the data for the prompt targeting both classes which are saved in the prompt log file
get_aug_tweet(context, prompt_content, oai_llm)	Generates a tweet based on the context, prompt and openAI LLM model being used
def get_batch_indices(start_index, end_index, batch_size, last_interval)	Creates and returns a dict used to manage batches of augmented tweets
generate_tweet_batch(…)	Generates a batch of augmented tweets from the LLM. Long list of parameters described in docstring.
write_aug_tweets(aug_tweets_dict, target_class, out_file)	Writes out a dictionary of generated tweets to a CSV file
get_random_samples(…)	Writes out a num_samples number of random samples drawn from df_data after creating a 'notes' and 'questionable_label' columns and filling the later with 2 (representing 'uncertain label'). Long list of parameters described in docstring.
count_tokens(text_vector, sort_ascending, special_tokens, vocabulary_size)	Counts the number of tokens in text_vector based on a set vocabulary size
get_tweet_path(…)	Builds and returns a string path to a batch of tweets. Long list of parameters described in docstring.
get_tweet_outfile(…)	Builds and returns a string path used to write a batch of tweets. Long list of parameters described in docstring.
consolidate_tweet_batches(…)	Reads in a set of files into dataframes, concatenates them vertically (by rows), writes out the consolidated file and returns it to the caller. Long list of parameters described in docstring.
get_vocabulary(path_to_vocab)	Reads in a vocabulary file and returns a set of words in the vocabulary
preproccess_pipeline(…)	Runs the entire preprocessing pipeline from the point immediately after the duplicates (both text-target and cross-target) have been removed. Long list of parameters described in docstring.
word_NN(w, vocab_embeddings, debug)	Finds the word closest to w in the vocabulary that isn't w itself.
vectorize_tweet(…)	Creates a one-dimensional vector which is the concatenation of each embedding in tweet_string padded to 30 tokens with pad_token. Long list of parameters described in docstring.

FUNCTION (cont.)	DESCRIPTION (cont.)
`make_tweet_feats(list_of_vectors)`	Builds and returns a 1 dimensional vector built by vertically stacking all the vectors in list_of_vectors
`get_roc_curves(…)`	Creates a plot of ROC curves and their corresponding AUC values for a set of models. Long list of parameters described in docstring.
`nn_train(…)`	Trains a neural network model over n_epochs using mini-batch gradient descent and returns the best accuracy. Long list of parameters described in docstring.

Appendix G – GloVe embedding for the empty string

On line 38523 of each of the twitter GloVe embeddings files is the vector for the empty string. The vector for this token could not be found by the **`get_glove_embeds`** function because there was no character present that the function could use to identify the vector to return. To remedy this, the two character token <> was inserted at the beginning of this line in a local copy of the file and served to identify the empty string.

The vector for the empty string was verified by the procedure described in my stackoverflow post here:

https://stackoverflow.com/questions/78283056/glove-embedding-for-empty-string/78285097#78285097

Appendix H – Special token frequencies

The 4 special tokens related to the Twitter @ and # characters as well as for URLs and numbers were the top 4 most frequent tokens for each class. The <url> token was the most frequent token in both classes, but the other 3 ranked differently by class as shown in Figures H1 through H3.

This difference in ranking could be a potentially useful feature in building a classifier designed for high accuracy.

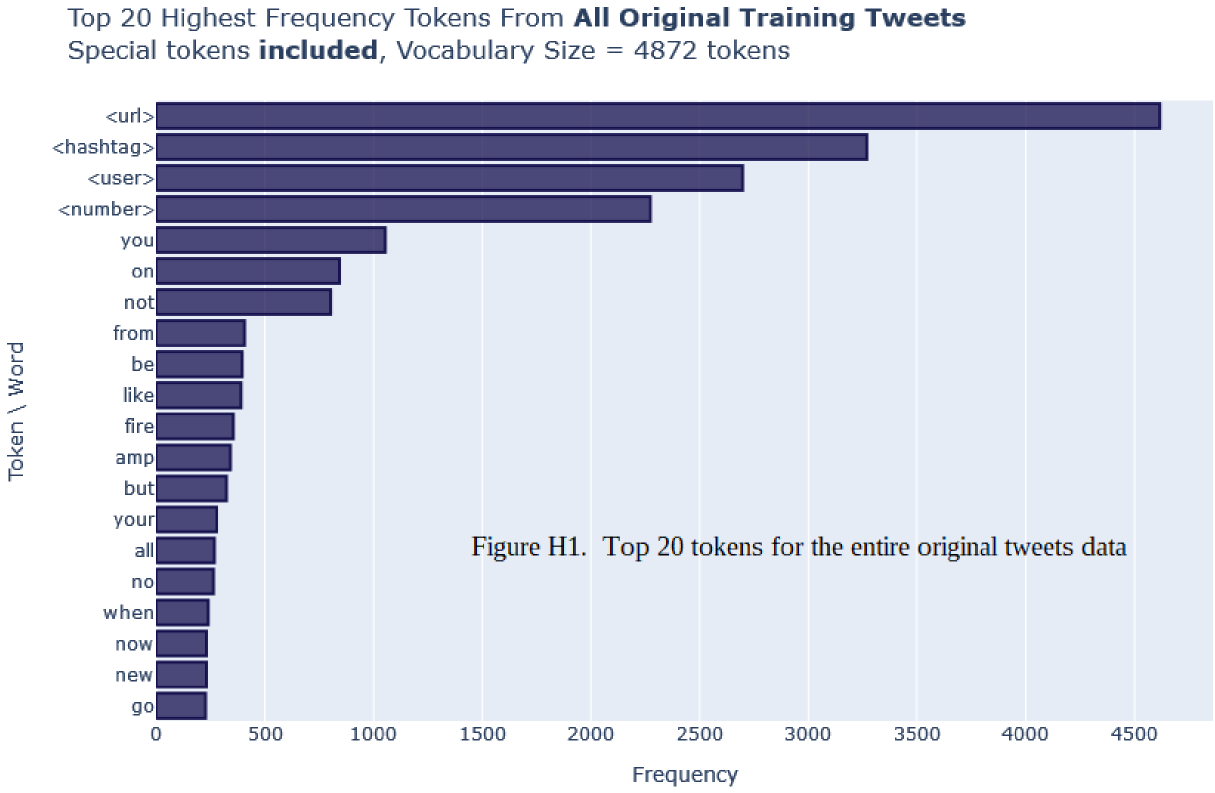


Figure H1. Top 20 tokens for the entire original tweets data

Appendix H (cont.)

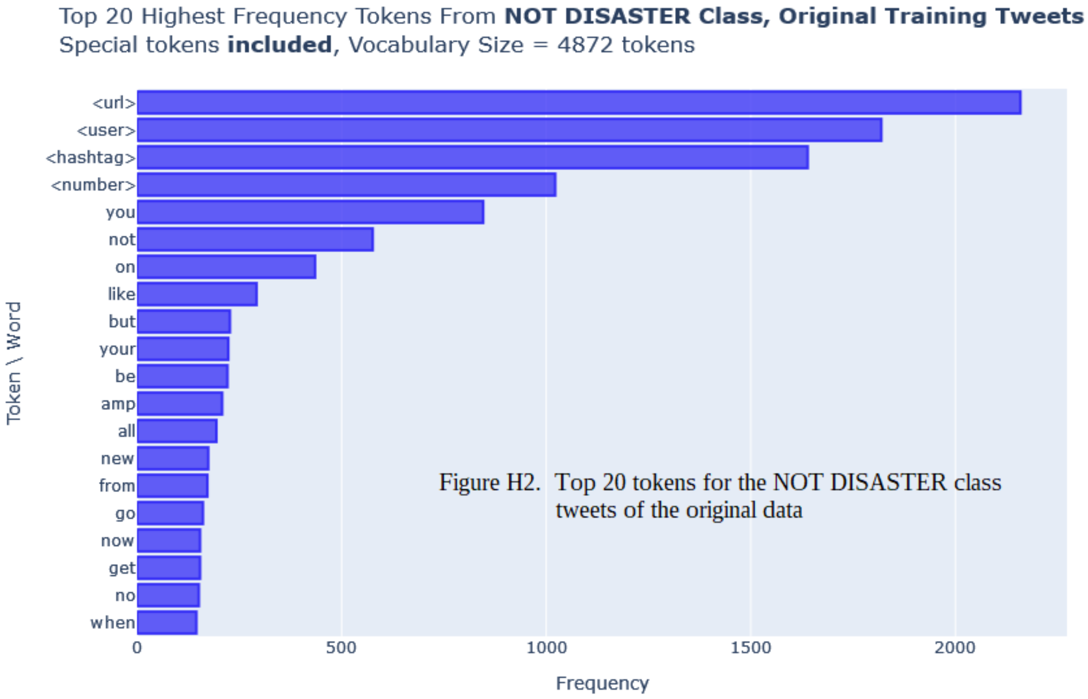


Figure H2. Top 20 tokens for the NOT DISASTER class tweets of the original data

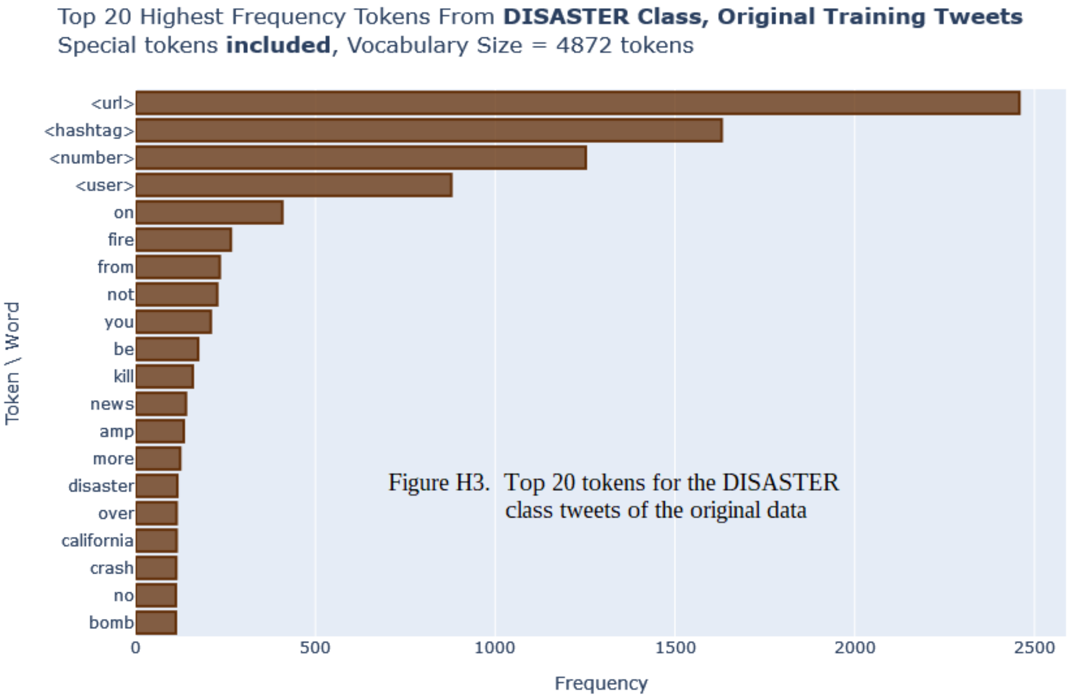


Figure H3. Top 20 tokens for the DISASTER class tweets of the original data

Appendix I – Prototypes of augmented data

I.1 Prompts for disaster-related tweets

The prompt designed to create the augmented data for the **disaster** class tweets went through the following iterations. The term “iprompt” refers to the instruction component of the prompt described in section 5.2:

Version	**Prompt Component**	**Content**
1	context	*You are a fiction writer who has observed a **disaster** and likes to tweet.*
1	iprompt	*Write me a tweet similar to this one, under 141 characters, but refers to a different **disaster** and location:*
2	context	**[same as Version 1]**
2	iprompt	*Write me a tweet similar to this one, under 141 characters, does not use contractions, but refers to a different **disaster** and location:*
3	context	**[same as Version 1]**
3	iprompt	*Write me a tweet similar to this one, under 141 characters, does not contain contractions or double quotes, but refers to a different **disaster** and location:*
4	context	*You are a world class reporter who has observed a **disaster** and likes to tweet.*
4	iprompt	*Write me a tweet similar to this one, under 141 characters, does not contain double quotes, but refers to a different **disaster** and location:*
5 (final)	context	**[same as Version 4]**
5 (final)	iprompt	*Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different **disaster** and location:*

The table shown on the following page are 4 examples of actual responses from ChatGPT 3.5 to complete version 5 prompts generated from the code ran in the **chatgpt_sandbox.ipynb** notebook.

Appendix I (cont.)

COMPLETE PROMPT	CHAT3.5 RESPONSE
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different disaster and location: *Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all*	*Witnessing the devastation caused by the powerful hurricane in the Caribbean region. Praying for the safety and well-being of all those affected. #hurricane #Caribbean*
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different disaster and location: *#Kurds trampling on Turkmen flag later set it ablaze while others vandalized offices of Turkmen Front in #Diyala http://t.co/4IzFdYC3cg*	*In #Earthquake aftermath in #Haiti, survivors desperate for aid as homes lie in ruins & hope fades. Devastation beyond words. #PrayForHaiti* 🙏🇭🇹
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different disaster and location: *The shooting or the airplane accident https://t.co/iECc1JDOub*	*Tragic scene here in Indonesia after a volcanic eruption at Mount Merapi. Thoughts and prayers go out to all those affected by this natural disaster.* 🌋 *#MountMerapi #Indonesia*
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different disaster and location: *But if you build an army of 100 dogs and their leader is a lion all dogs will fight like a lion.*	*In the aftermath of the earthquake in Nepal, communities are coming together as one to rebuild their homes and lives #NepalStrong* 🌏🙏

Appendix I (cont.)

I.2 Prompts for NOT-disaster-related tweets

The prompt designed to create the augmented data for the **not disaster** class tweets went through the following iterations. The same context was used for both classes in the first version. The term "iprompt" refers to the instruction component of the prompt described in section 5.2:

Version	Prompt Component	Content
1	context	*You are a fiction writer who has observed a* ***disaster*** *and likes to tweet.*
1	iprompt	*Write me a tweet similar to this one, under 141 characters, but refers to a different* ***activity*** *and location:*
2	context	**[same as Version 1]**
2	iprompt	*Write me a tweet similar to this one, under 141 characters, does not use contractions, but refers to a different* ***activity*** *and location:*
3	context	**[same as Version 1]**
3	iprompt	*Write me a tweet similar to this one, under 141 characters, does not contain contractions or double quotes, but refers to a different* ***activity*** *and location:*
4	context	*You are a social commentator who enjoys diverse opinions and likes to tweet.*
4	iprompt	*Write me a tweet similar to this one, under 141 characters, does not contain double quotes, but refers to a different* ***activity****,* ***feeling*** *and location:*
5 (final)	context	**[same as Version 4]**
5 (final)	iprompt	*Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different* ***activity****,* ***feeling*** *and location:*

The table shown on the following page are 4 examples of actual responses from ChatGPT 3.5 to complete version 5 prompts generated from the code ran in the **chatgpt_sandbox.ipynb** notebook.

Appendix I (cont.)

PROMPT	CHAT3.5 RESPONSE
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different activity, feeling and location: *What's up man?*	*Hey there! How are you feeling after that intense workout at the gym? 💪🏋️ #fitnessgoals #gymlife*
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different activity, feeling and location: *Summer is lovely*	*Oh hayrides in the fall, cozy sweaters and pumpkin spice everything, autumn vibes are here 🍂🍁 #seasons #fallvibes*
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different activity, feeling and location: *Was in NYC last week!*	*Just had the most amazing sushi in Tokyo! 🍣 Can't get over the flavors and ambiance here. #TravelAdventures #FoodieMoments*
Write me a tweet similar to this one in length and content, under 141 characters, does not contain double quotes but refers to a different activity, feeling and location: *Crying out for more! Set me ablaze*	*Thriving for connection! Ignite my soul 🔥 #passionate #community*

Appendix J – Three main ways to vectorize text

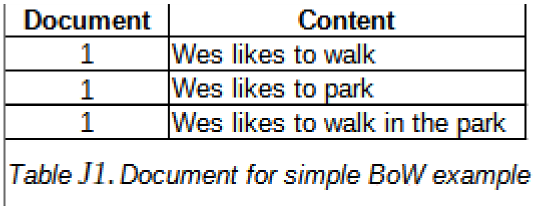
J.1 One-Hot Encoding and Bag-of-Words

The simplest way to do word-level encoding is with one-hot encoding. In this method, a binary vector the size of the vocabulary $\mathcal{V}$ is assigned to each word [12]. Each position in this $\mathcal{V}$-dimensional vector corresponds to a word in the vocabulary. A word is represented by putting a 1 in the location corresponding to that word and 0's in the remaining locations.

One-hot encoding is extended to documents through the bag of words (BoW) model [13]. In this representation, each document is simply the sum of all the one-hot encoded word vectors in the document. For example, let's say we had the following 7 word vocabulary [14]:

Wes likes to walk in the park

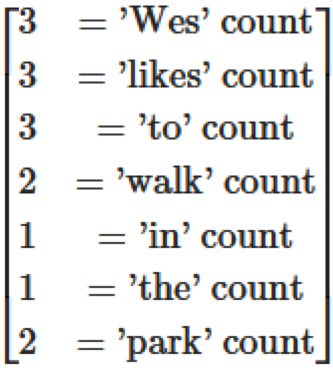
and a document with the following 3 sentences as shown in Table J1.



Document	Content
1	Wes likes to walk
1	Wes likes to park
1	Wes likes to walk in the park

Table J1. Document for simple BoW example

The BoW representation of this document would look like what is shown in Figure J1. While this is a simple solution to the vectorization problem, it has two notable issues. The first problem is that the size of these vectors are typically quite large. For example, an average 20-year old native speaker of American English is estimated to have a vocabulary between 27k and 52k words [15].



$$\begin{bmatrix} 3 & = \text{'Wes' count} \\ 3 & = \text{'likes' count} \\ 3 & = \text{'to' count} \\ 2 & = \text{'walk' count} \\ 1 & = \text{'in' count} \\ 1 & = \text{'the' count} \\ 2 & = \text{'park' count} \end{bmatrix}$$

Figure J1. Simple BoW example

The second problem is that no information about the meaning of words or the relationship between words is captured in this representation. The vector in Figure J1. could be reordered in any way and it would convey the same information as long as whatever order is chosen is maintained.

J.2 TF–IDF

Term Frequency – Inverse Document Frequency improves on simple word counts by creating a statistic which measures how frequently a word occurs within a document relative to across documents. Words that occur within a document but not across documents are considered more interesting and score higher than words that occur frequently with and across documents [16]. While this improves upon simple word counts, meaning of words and their relationships to each other are still not captured using this approach.

J.3 Word Embeddings

Word embeddings address the two biggest problems with one-hot encoded words by using much smaller vectors containing real values as compared to integer values resulting from simply counting words. Instead of vectors that are tens of thousands of dimensions, word embeddings such as GloVe typically range between 25 and 300 dimensions [17].

Besides being much smaller, word embeddings capture some of the semantic relationships between words. For example, a gender relationship could be expressed as the difference between two pairs of vectors such as: king and queen, man and woman, sir and madam. In other words, if our representation captures relationships between genders, we would expect the following equation to hold true:

$$\overline{V}_{gender} \approx \overline{V}_{king} - \overline{V}_{queen} \approx \overline{V}_{man} - \overline{V}_{woman} \approx \overline{V}_{sir} - \overline{V}_{madam} \quad \text{(J.1)}$$

where $\overline{V}_w$ is the word embedding vector representation for word w. In fact, this is what we see for GloVe embeddings as shown in the figure on the right.

Figure J2. is taken from the GloVe website [17] and plots the first two principal components of GloVe word embeddings for several word pairs with a gender relationship. The dashed lines represent a 2-dimensional projection of the difference between the two words.

The similarity between these pairs of vectors can be quantified using cosine similarity [18] and approximated visually by the slope of the dashed line. Notice how these lines are all oriented roughly vertically and slope slightly to the right. This type of consistency is what we would expect from equation (J.1).

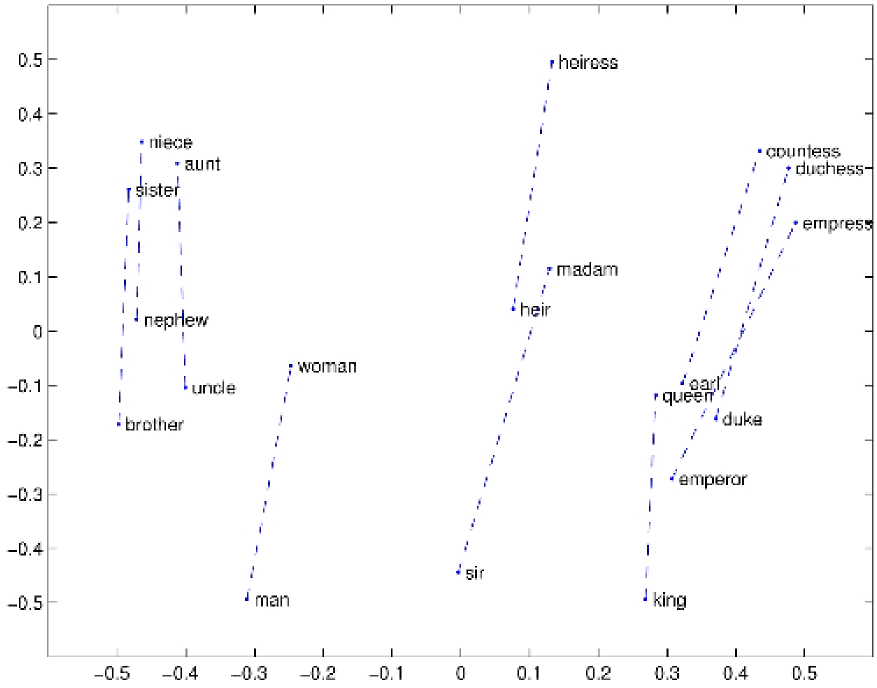


Figure J2. Word pairs with gender relationship

Because word embeddings are far superior to one-hot encoded vectors, they were chosen as the encoding method for this project. After choosing the general word encoding scheme, the choice of which embeddings to use and how to encode documents also had to be made. Word embeddings come in two major types. The first type are referred to as custom embeddings which are computed from a domain-specific corpus of text. The second type are pre-trained or “canned” embeddings which have been computed in advance from a generic or/and publicly available corpus of text.

J.3 Word Embeddings (cont.)

Custom embeddings typically perform better than pre-trained embeddings on NLP classification tasks because they are more likely to capture subtle distinctions between a specific target domain (e.g. biomedical, economic or other research areas) and general public discourse. Because this project focused on classifying publicly available tweet data, the additional work required to build custom embeddings could not be justified and was dropped from consideration.

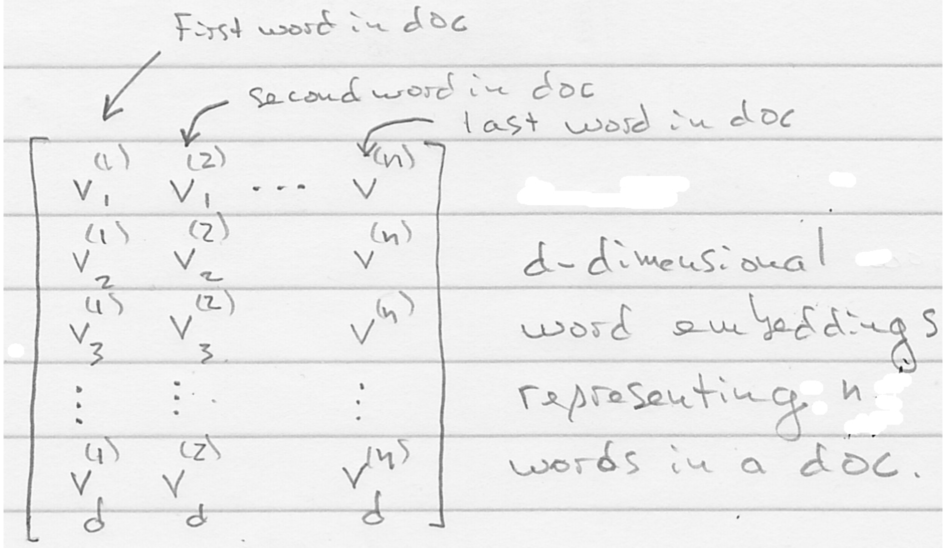
A number of pre-trained word embeddings are available for use. Two of the more popular are Global Vectors or GloVe developed at Stanford University and word2vec developed by Google. GloVe embeddings are available in several dimensions (25d, 50d, 100d, 200d and 300d) as well as type determined by the data that they were trained on such as: Wikipedia 2014 + Gigaword 5, Common Crawl and Twitter [800]. Only one set of 300d pre-trained word2vec embeddings which were trained from a GoogleNews corpus are available [19]. It is difficult to assess which of these two are superior, but the existing evidence suggests that GloVe performs comparable to word2vec [9, page 9].

Pre-trained GloVe embeddings were selected for this project because they perform similarly to word2vec and more importantly because a twitter-specific embeddings existed in a choice of various dimensions. This last characteristic strongly implied that some the benefits of a custom embedding could be gained without having to do the extra work of training on another corpus.

Appendix K – Creating a document vector from embedding encoded words

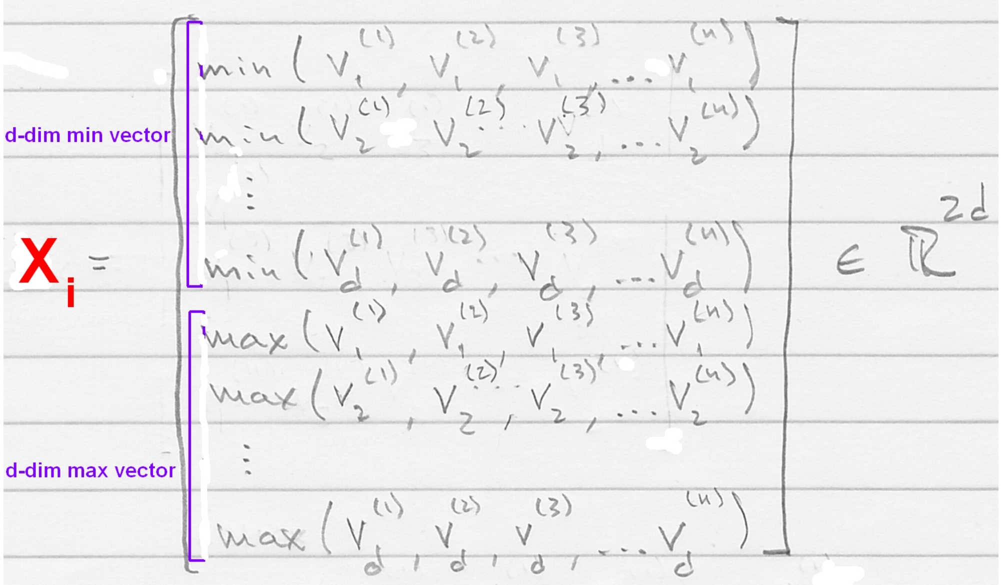
K.1 Using min, max and mean vectors to represent documents as a single vector

If we wrote each d-dimensional word embedding in a document of n words as a column vector and stacked them horizontally, we would form the following d x n matrix:


First word in doc
second word in doc
last word in doc
$V_1^{(1)}$ $V_1^{(2)}$ . . . $V^{(n)}$
$V_2^{(1)}$ $V_2^{(2)}$ $V^{(n)}$
$V_3^{(1)}$ $V_3^{(2)}$ $V^{(n)}$
$V_d^{(1)}$ $V_d^{(2)}$ $V_d^{(n)}$
d-dimensional word embeddings representing n words in a doc.

The drawing on the right depicts representing a document as a matrix, but what we need is a representation of a document as a **vector**. We can do this by collapsing all columns in this matrix into a single column. One way to do this would be to take the **coordinate minimums** which are the minimums of each row. We can then do the same, but take **coordinate maximums** and then stacking them on top of each other. When we do this, each document becomes a fixed length vector that looks like the drawing shown at the bottom of the page.

Other variations on this theme are to use the mean instead of the min and max to create a vector that is d-dimensions or to add the mean to the stack shown in drawing below to create a vector that is 3 x d dimensions in length.


X_i =
d-dim min vector
d-dim max vector
$\min(V_1^{(1)}, V_1^{(2)}, V_1^{(3)}, \ldots V_1^{(n)})$
$\min(V_2^{(1)}, V_2^{(2)}, V_2^{(3)}, \ldots V_2^{(n)})$
$\min(V_d^{(1)}, V_d^{(2)}, V_d^{(3)}, \ldots V_d^{(n)})$
$\max(V_1^{(1)}, V_1^{(2)}, V_1^{(3)}, \ldots V_1^{(n)})$
$\max(V_2^{(1)}, V_2^{(2)}, V_2^{(3)}, \ldots V_2^{(n)})$
$\max(V_d^{(1)}, V_d^{(2)}, V_d^{(3)}, \ldots V_d^{(n)})$
$\in \mathbb{R}^{2d}$

K.2 Using the first n token vectors to represent documents as a single vector

Instead of using the min, max or mean as described in the previous section, we could just stack the first n word vectors to represent a document. This approach is practical if a good value for n is chosen. Because each tweet is less than 30 tokens, we chose n = 30 which captures all the tokens in each pre-processed tweet and pads the remaining space with the empty string embedding.

For example, say we have a tweet: ***`summer is lovely`*** and we are using 50d twitter glove embeddings, each word would be represented by the following vectors where ... are the values for the other 45 dimensions in the 50d vector:

summer = [-0.40501, -0.56994, 0.34398, ..., -0.95337, 1.1409] is = [0.18667 0.21368 0.14993, ..., -0.24608, -0.19549] lovely = [-0.27926 -0.16338 0.50486, ..., -0.15416, -0.20196]

If we concatenate these three word vectors, we get 150d vector because it only had 3 tokens. We then concatenate 27 empty string vectors which look like this:

<> = [-0.32053, -0.73053, -0.15227, … 0.73164, -0.23074]

to bring the vector size up to 1500 dimensions.

Another advantage of using all the token vectors in sequence to represent the document, is that this representation provides a way for the neural network to learn features related to how words are used in sequence without having to resort to more complex architectures such as convolutional or recurrent networks.

Appendix L – Results of 10-fold cross-validation for hidden-layer activation function

Two runs were done spaced two days apart and similar results were obtained as shown in the table below. ReLU showed slightly better results and is more computationally efficient, so it was chosen as the hidden-layer activation function.

2024-04-12: 10-fold CV results of ReLU vs Sigmoid activation	**2024-04-14: 10-fold CV results of ReLU vs Sigmoid activation**
ReLU accuracy: 0.7786	ReLU accuracy: 0.7762
ReLU accuracy: 0.7971	ReLU accuracy: 0.9737
ReLU accuracy: 0.8186	ReLU accuracy: 0.9761
ReLU accuracy: 0.8115	ReLU accuracy: 0.9523
ReLU accuracy: 0.8568	ReLU accuracy: 0.9594
ReLU accuracy: 0.8282	ReLU accuracy: 0.9809
ReLU accuracy: 0.8162	ReLU accuracy: 0.9833
ReLU accuracy: 0.8783	ReLU accuracy: 0.9761
ReLU accuracy: 0.8616	ReLU accuracy: 0.9857
ReLU accuracy: 0.9165	ReLU accuracy: 0.9857
ReLU mean acc: 83.63% std dev: 3.94%	**ReLU mean acc: 95.49**% std dev: 6.05%
sigmoid accuracy: 0.7762	sigmoid accuracy: 0.7619
sigmoid accuracy: 0.8067	sigmoid accuracy: 0.8258
sigmoid accuracy: 0.7971	sigmoid accuracy: 0.8019
sigmoid accuracy: 0.7661	sigmoid accuracy: 0.8282
sigmoid accuracy: 0.8401	sigmoid accuracy: 0.8377
sigmoid accuracy: 0.8091	sigmoid accuracy: 0.8449
sigmoid accuracy: 0.8544	sigmoid accuracy: 0.8616
sigmoid accuracy: 0.8377	sigmoid accuracy: 0.8282
sigmoid accuracy: 0.8282	sigmoid accuracy: 0.8687
sigmoid accuracy: 0.8377	sigmoid accuracy: 0.8568
sigmoid mean acc: 81.53% std dev: 2.78%	sigmoid mean acc: 83.16% std dev: 2.99%

Appendix M – Results of 10-fold cross-validation for number of nodes in hidden-layer

Two runs were done spaced two days apart and similar results were obtained as shown in the table below. Results of the first run indicated that 300 units would be slightly better but the difference was much smaller than the standard deviation. Results of the second run showed better performance with 100 units. Because no clear winner emerged, 100 units was chosen to make network training less demanding.

2024-04-12: 10-fold CV results of 100 vs. 300 hidden layer units & ReLU activation function	**2024-04-14: 10-fold CV results of 100 vs. 300 hidden layer units & ReLU activation function**
ReLU 100 hidden units - accuracy: 0.7738	ReLU 100 hidden units - accuracy: 0.8071
ReLU 100 hidden units - accuracy: 0.8568	ReLU 100 hidden units - accuracy: 0.8282
ReLU 100 hidden units - accuracy: 0.8544	ReLU 100 hidden units - accuracy: 0.8496
ReLU 100 hidden units - accuracy: 0.8019	ReLU 100 hidden units - accuracy: 0.8305
ReLU 100 hidden units - accuracy: 0.9332	ReLU 100 hidden units - accuracy: 0.8449
ReLU 100 hidden units - accuracy: 0.9523	ReLU 100 hidden units - accuracy: 0.8234
ReLU 100 hidden units - accuracy: 0.9451	ReLU 100 hidden units - accuracy: 0.8353
ReLU 100 hidden units - accuracy: 0.9403	ReLU 100 hidden units - accuracy: 0.8520
ReLU 100 hidden units - accuracy: 0.9117	ReLU 100 hidden units - accuracy: 0.8377
ReLU 100 hidden units - accuracy: 0.9379	ReLU 100 hidden units - accuracy: 0.8568
ReLU with 100 hidden units mean accuracy: 89.07% std dev: 6.14%	ReLU with 100 hidden units **mean accuracy: 83.66%** std dev: 1.43%
ReLU 300 hidden units - accuracy: 0.7952	ReLU 300 hidden units - accuracy: 0.8190
ReLU 300 hidden units - accuracy: 0.8807	ReLU 300 hidden units - accuracy: 0.7852
ReLU 300 hidden units - accuracy: 0.9236	ReLU 300 hidden units - accuracy: 0.8544
ReLU 300 hidden units - accuracy: 0.9045	ReLU 300 hidden units - accuracy: 0.8162
ReLU 300 hidden units - accuracy: 0.9021	ReLU 300 hidden units - accuracy: 0.7780
ReLU 300 hidden units - accuracy: 0.9141	ReLU 300 hidden units - accuracy: 0.8043
ReLU 300 hidden units - accuracy: 0.9451	ReLU 300 hidden units - accuracy: 0.8401
ReLU 300 hidden units - accuracy: 0.9093	ReLU 300 hidden units - accuracy: 0.8305
ReLU 300 hidden units - accuracy: 0.8950	ReLU 300 hidden units - accuracy: 0.8377
ReLU 300 hidden units - accuracy: 0.9021	ReLU 300 hidden units - accuracy: 0.8711
ReLU with 300 hidden units **mean accuracy: 89.72%** std dev: 3.77%	ReLU with 300 hidden units overall accuracy: 82.37% std dev: 2.78%

Appendix N – Label errors in kaggle training data

A sample of 400 class 0 and 400 class 1 samples were taken to quantify the extent of labeling errors in the kaggle Disaster Tweets dataset. The table below summarizes these results listed in:

https://ln5.sync.com/dl/580072ff0/ni4nuiji-cbxgrww3-2pq7riwz-nb5q95ts

target class	judged correct	judged incorrect	unsure
0	391	2	7
1	264	96	40

Because a large majority of the NOT DISASTER tweets appeared to be labeled correctly, it was assumed these tweets presented no major problems for the models. The DISASTER tweets on the other had were a different story. The table below shows a few examples of what the author would consider labeling errors in the training data. Out of sample of 400 disaster tweets, 96 were identified as mislabeled. The **target** column is the class label where 0 = NOT DISASTER and 1 = DISASTER.

id	**text**	**target**	**notes**
4882	Kendall Jenner and Nick Jonas Are Dating and the World Might Quite Literally Explode http://t.co/pfvzVPxQGr	1	two celebs dating
8880	I get to smoke my shit in peace	1	probably not explosive
4142	We happily support mydrought a project bringing awareness to the LA drought. Track your water⑨Û_ https://t.co/2ZvhX41I9v	1	support for a disaster project ≠ disaster
4973	ITS A TIE DYE EXPLOSION ON IG HELP ME. IM DROWNING IN TIE DYE	1	likely only a problem for the textile artist

The table below shows a couple examples of tweets that the author was unsure what the class was:

id	**text**	**target**	**notes**
5356	It may seem like our fire has been a little burnt out....	0	word sense is not clear here
9077	@sirtophamhat @SCynic1 @NafeezAhmed @jeremyduns and of course you don't have to melt the steel in order to cause structural failure.	0	could have been referencing an active fire
3066	500 deaths a year from foodborne illness... @frackfreelancs dears... @DECCgovuk @frackfree_eu @tarleton_sophie http://t.co/JSccX8k0jA	1	ambiguous reference
6187	Governor allows parole for California school bus hijacker who kidnapped 26 children in 1976. http://t.co/hdAhLgrprl http://t.co/Z1s3T77P3L	1	don’t see how this can be considered a disaster

Appendix O – Earlier approaches to data augmentation

O.1 Back Translation

Because Back Translation (BT) requires access to a language translation model (e.g. Google Translate), this approach can be viewed as a crude approximation of the technique being investigated by this project. The main difference is the degree to which the output of the augmentation process can be influenced. With BT, the user has very little influence on the output other than selecting the intermediate language to translate the original sentence or phrase into. Even when intermediate languages are changed, the output can come back the same as the input. When this happens, the generated sample is thrown out. Another artifact of BT is that many of the generated samples may not differ enough to provide models with enough relevant information to improve their performance.

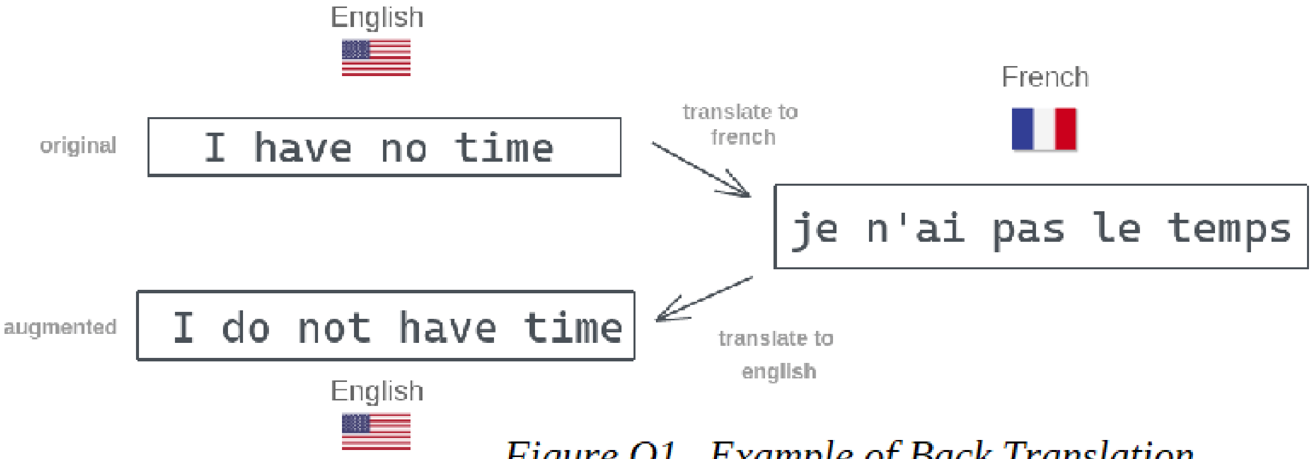


Figure O1. Example of Back Translation

O.2 Easy Data Augmentation (EDA)

A simpler approach to augmentation than BT are a set of transformations collectively referred to as *Easy Data Augmentation* or EDA [8] (not to be confused with Exploratory Data Analysis). The four transformations that make up EDA are:

1. **Synonym Replacement**: Randomly choose *n* words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.
2. **Random Insertion:** Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. Do this *n* times.
3. **Random Swap:** Randomly choose two words in the sentence and swap their positions. Do this *n* times.
4. **Random Deletion:** Randomly remove each word in the sentence with probability *p*.

The authors of this technique present compelling evidence that the augmented sentences maintain their class labels and generally boost performance of models trained with EDA augmented data. This suggests that this approach is superior to BT in terms of intuitiveness, ease of implementation and improved training on models that include this kind of augmented data.

O.3 Transformers

Transformer architecture is at the heart of what empowers LLMs such as BERT and ChatGPT. They were designed to implement more efficient sequence transduction models [21, 22]. In the context of NLP, sequence transduction models predict the next word/token directly from the data instead from a function derived from the data which is the typical supervised learning use case. The k-Nearest Neighbors (KNN) model is probably the most well know example of a transduction model.

Transformers rely on positional encoding and a neural network architectural sub-component know as the attention mechanism. From the perspective of the transformer, documents are long sequences of words or tokens. Positional encoding provides information that allows the attention mechanism to learn which words are related to others in the sequence. It is this ability to learn the relationship between words in a document that allows an LLM to generate human-like abilities in language-related tasks.

Although some would argue that BERT is more powerful than ChatGPT because it processes text bi-directionally [23], ChatGPT 3.5 turbo was selected as the transformer over BERT primarily out of convenience. The ChatGPT API is easy to use and tokens are relatively cheap. BERT requires pre-training and fine-tuning steps before it can be used for NLP tasks.

References

[1] https://www.turing.com/kb/introduction-to-self-supervised-learning-in-nlp#supervised-learning

[2] Bengio, Y., Courville, A., & Vincent, P. (n.d.). *Representation learning: A review and new perspectives*. Arxiv.org. Retrieved April 18, 2024, from http://arxiv.org/abs/1206.5538

[3] Ariz, K. (2023, March 16). *What are the most common NLP tasks?* Scalo. https://carttune.ai/what-are-the-most-common-nlp-tasks/

[4] (N.d.-c). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[5] Shahul, E. S. (2022, July 21). *Data augmentation in NLP: Best practices from a Kaggle master*. Neptune.Ai. https://neptune.ai/blog/data-augmentation-nlp

[6] Feng, S. Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., & Hovy, E. (n.d.). A survey of data augmentation approaches for NLP. Arxiv.org. Retrieved March 18, 2024, from http://arxiv.org/abs/2105.03075

[7] Chaudhary, A. (2020, February 20). Back translation for text augmentation with Google Sheets. Amit Chaudhary. https://amitness.com/2020/02/back-translation-in-google-sheets/

[8] Wei, J., & Zou, K. (2019). EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In arXiv [cs.CL]. http://arxiv.org/abs/1901.11196

[9] Pennington, J., Socher, R., & Manning, C. D. (n.d.). GloVe: Global vectors for word representation. Stanford.edu. Retrieved March 19, 2024, from https://nlp.stanford.edu/pubs/glove.pdf

[10] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2022). An Introduction to Statistical Learning: with Applications in R (2nd ed.). Springer.

[11] Desell, Travis “Lecture 1 Feed Forward Neural Network Design”, DSCI 640, Spring 2023. Rochester Institute of Technology. Lecture (N.d.-f). Retrieved March 19, 2024, from https://github.com/MichaelSzczepaniak/llmamd/blob/main/docs/lectures_dsci_640_1_thru_8.pdf

[12] (N.d.-c). Machinelearningmastery.com. Retrieved March 18, 2024, from https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[13] (N.d.-d). Machinelearningmastery.com. Retrieved March 18, 2024, from https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[14] Szczepaniak, M. (n.d.). Word Embeddings. Retrieved March 18, 2024, from https://github.com/MichaelSzczepaniak/WordEmbeddings/blob/master/WordEmbeddings.ipynb

[15] Brysbaert, M., Stevens, M., Mandera, P., & Keuleers, E. (2016). How many words do we know? Practical estimates of vocabulary size dependent on word definition, the degree of language input and the participant’s age. Frontiers in Psychology, 7. https://doi.org/10.3389/fpsyg.2016.01116

[16] (N.d.). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

[17] Pennington, J. (n.d.). GloVe: Global Vectors for word representation. Stanford.edu. Retrieved March 19, 2024, from https://nlp.stanford.edu/projects/glove/

[18] (N.d.-b). Baeldung.com. Retrieved March 19, 2024, from https://www.baeldung.com/cs/cosine-similarity

[19] GoogleNews-vectors-negative300.bin.gz. (n.d.). Google Docs. Retrieved March 19, 2024, from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing

[20] https://stackoverflow.com/questions/1787110/what-is-the-difference-between-lemmatization-vs-stemming

[21] (N.d.-d). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/transduction-in-machine-learning/

[22] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In arXiv [cs.CL]. http://arxiv.org/abs/1706.03762

[23] Budhathoki, S. (2023, March 9). ChatGPT vs. BERT: Battle of the transformer. The Nature Hero. https://thenaturehero.com/chatgpt-vs-bert/