# Using a Large Language Model to Improve the Performance of Simpler NLP Classifier Models

## 1. Abstract

Recent advances in the fields of computational linguistics and neural network architectures have enabled dramatic improvements in the performance of most natural language processing (NLP) tasks such as text classification. However, these improvements are limited by the fundamental challenge of having enough labeled data to train these models [g]. This project investigates an approach to this problem that utilizes the publicly available large language model (LLM), ChatGPT 3.5 turbo to augment the kaggle Disaster Tweets data set to do binary classification [d]. The viability of this approach is 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.

## 2. Background and Introduction

Word embeddings, transformer architecture and prompt engineering have become important themes in the story behind the rise of LLMs. Word embeddings are used in vectorizing text to a much more efficient and information-rich representation from which NLP models can learn compared to vectors constructed from counting word occurrences [1]. Transformer architecture allows NLP models to more accurately learn which parts of text refer to other parts of text in a document [2]. Prompt engineering becomes important when an LLM becomes available for use and focuses on providing inputs to that LLM which leads to obtaining meaningful results [h].

### 2.1 The Vectorization Problem

#### 2.1.1 One-Hot Encoding and Bag-of-Words

Before any model using text as input can be trained, that text must be converted to a vector. There are a number of ways to do this vectorization, but the simplest way to do this at the word-level is with one-hot encoding which assigns a binary vector the size of the vocabulary $\mathcal{V}$ to each word [4]. 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 typically extended to documents through the bag of words (BoW) model [i]. 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 [5]:

*Wes likes to walk in the park*

and a document with the following 3 sentences:

*Wes likes to walk*
*Wes likes to park*
*Wes likes to walk in the park*

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

*Figure 1. Simple BoW example*

The BoW representation of this document would look like what is shown in Figure 1.

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 [6].

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 1. could be reordered in any way and it would convey the same information as long as whatever order is chosen is maintained.

### 2.1.2 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 [7] typically range between 25 and 300 dimensions.

Besides being much smaller, word embeddings capture some of the 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:

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

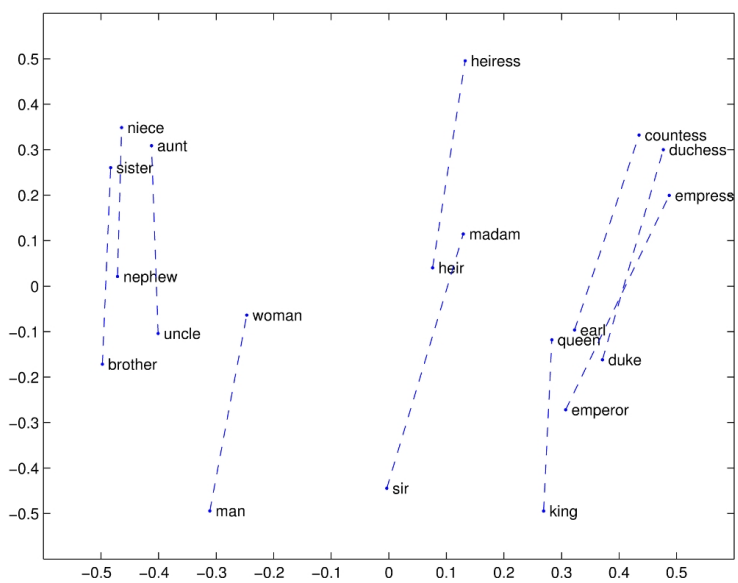


*Figure 2. Word pairs with gender relationship*

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 left figure.

Figure 2. is taken from the GloVe website [7] 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 [8, 9] 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 (1).

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 must also 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.

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 [10]. 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 embedding are available for use. Two of 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 (Wikipedia 2014 + Gigaword 5, Common Crawl and Twitter). Only one set of 300d pre-trained word2vec embeddings which were trained from a GoogleNews corpus are available [s]. It is difficult to assess which of these two are superior, but the existing evidence suggests that GloVe may be slightly better [7b].

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.

## 2.2 Transformer Architecture

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 [a, b]. 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 sequence 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 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 [t], 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.

## 2.3 Prompt Engineering

Prompts are essentially the inputs supplied to generative AI models such as ChatGPT, Stable Diffusion, et. al.. Prompts can be questions or commands and can include longer additional statements to provide context, instructions or other relevant information that users want the AI model to consider. The quality of AI model outputs is strongly determined by the prompt. For example, consider the following two prompts [j]:

1) *Write a short text explaining prompt engineering.* and
2) *Write a two sentence text explaining prompt engineering.*

The response to the first prompt from ChatGPT might be something similar to the following:

*Prompt engineering refers to crafting effective and specific instructions or prompts when interacting with language models like ChatGPT. It involves carefully designing the input to elicit desired responses and obtain the desired output. Prompt engineering can help shape the model's behavior, improve the quality of generated responses, and enhance the overall user experience by providing clear guidelines, constraints, or examples.*

The response to the second prompt from ChatGPT might be something similar to the following:

*Prompt engineering involves crafting effective and specific instructions for language models like ChatGPT to shape their behavior and improve user experience by providing clear input guidelines. It aims to elicit desired responses and enhance output quality.*

As this example shows, the degree of precision used in the prompt will impact the kind of output that is received. For this reason, an iterative process of trial-and-error is required to converge upon an effective prompt. This process can be made more efficient by following these guidelines [k]:

- Start simple -- This let's you know if you in general area of what you are looking for.
- Be specific -- This increase the likelihood of getting desired outcomes.
- Avoid Impreciseness -- In the previous example, the first prompt was less precise.
- Avoid say what not to do -- Focus on what the model is to do results in better results overall.

## 3. 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 are available [l] 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 [g].

In the domain of NLP-related tasks, many data augmentation techniques have been developed [r]. Three which will be described here are Back Translation (BT), Easy Data Augmentation (EDA) and NLP Albumentation (NLPA). BT generates a new word or phrase by first translating that text to a foreign language and then translating the foreign translation back to the original language as shown in Figure 3. [n]. EDA is a set of simple and intuitive transformations which can be applied to the original text data to

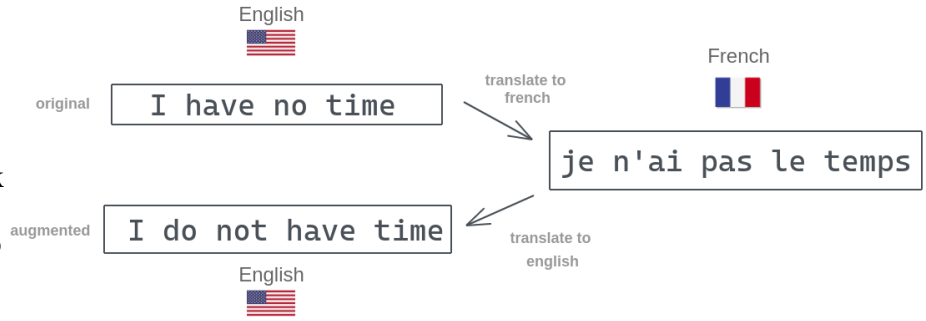


*Figure 3. Example of Back Translation*

generate augmented samples [o]. NLPA refers to a more complex set of transformations that were originally designed for computer vision but adapted to the NLP domain [p1, p2].

### 3.1 Back Translation

Because Back Translation (BT) requires access to a language translation model (e.g. Goolge 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 of 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.

### 3.2 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 [q]. 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.

### 3.3 NLP Albumentation

Adapted from the computer vision data augmentation domain [p1], NLP Albumentation (NLPA) applies a different set of transformations than EDA. There a numerous transforms described in this kaggle notebook [p2]. They all appear to be less intuitive then EDA and generally work by probabilistically shuffling or removing sentences from each sample.

Because is is unclear which transforms one should use under a given set of conditions, a lot of experimentation would be required to make effective use of this approach. For this reason coupled with it’s lack of intuitiveness, NLPA is currently considered to be less effective than EDA.

## 4. Text Classification Models

There are a number of choices for text classification models. Three types of these models will be considered here: Logistic Regression, Tree-based methods and Neural Networks. Logistic Regression is a generalized linear model which learns a probability that a sample is in each of the target classes based on the frequency of words (if one-hot encoding is used) or word-like features (if word

embeddings are used). Tree-base methods range in complexity from simple decision trees to more more sophisticated methods such as Random Forests. Neural Networks are class of non-linear functions consisting of numerous architectures and hyper-parameters designed to address particular aspects of the learning task it is being used for.

### 4.1 Logistic Regression

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:

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

where p(X) = P( y=1 | X ) and X are the p predictors [u1]. The β coefficients are typically determined by maximizing the likelihood function as described here [u2].

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 influence 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.

### 4.2 Tree-based methods

Tree-base models range from simple decision trees to more sophisticated methods such as boosted trees and random forest (RF) models. Decision trees partition the predictor space such that they minimize either a Gini index or an entropy function which rewards assigning higher probabilities to the correct class and lower probabilities to the incorrect class as described here [u3]. Although these models are intuitive and interpretable, they partition the predictor space in a staircase manner resulting in less predictive power than other classification approaches [u4].

The performance of tree-based models can be substantially enhanced by building a number of trees and aggregating their results. For example, random forest models build a “forest” of trees where each tree in the forest is built by dropping a random subset of its predictors before building the tree. Random forest models generally work very well, however, there are two meta-parameters which must be specified: n = number of trees in the forest and m = number of predictors to use in building trees in the forest. The rule of thumb (heuristic) for m is to set it equal to the square root of p where p = number of predictors in the dataset. While this works well in most use cases, the value for n must still be determined through some iterative process such as cross-validation.

A random forest model would have been a reasonable choice for the second model in this project, however they were eliminated from consideration for two reasons. RF models aren't normally used for text classification because they are better suited for tabular data and don't generally handle large number of predictors as well as neural networks. In addition to this limitation, at least one meta-parameter (n) needs to be determined.

## 4.3 Neural Networks

Neural Networks (NNs) have become the most 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.

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.

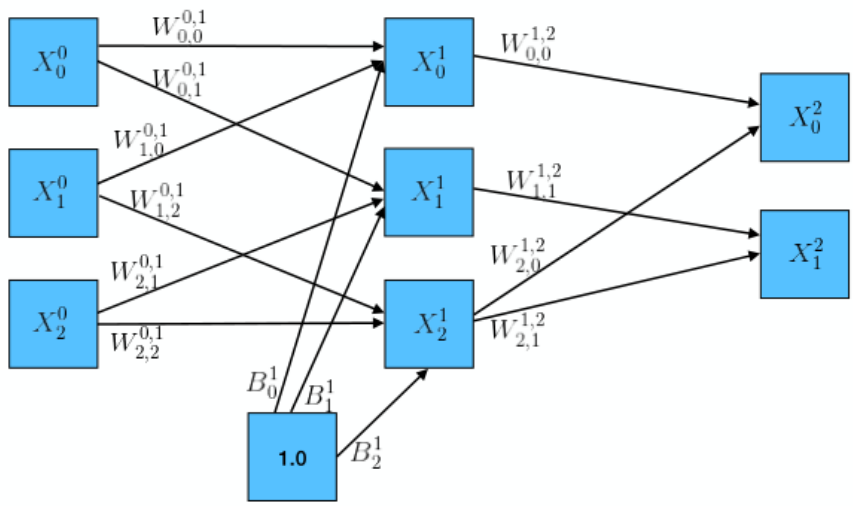


*Figure 4. Neural network with single hidden layer [v1]. Biases ($B^i_j$) not added to input or output nodes.*

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 4. 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 values of the hidden units are computed by first calculating the sum of each input coming 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 5.

There are many choices for activation functions, each having their own set of pros and cons as described here [v3]. After selecting an activation function for each layer however, additional design choices must be made before an NN-based model can be implemented. Other choices include:

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

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

*Figure 5. Calculation of node outputs of Figure 4. network*

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

Once the network structure has been fully specified, the weights and biases can be learned in a supervised fashion by following this general workflow [v4]:

1. Specifying a loss function appropriate for the task the network is being used for (e.g. classification, regression, etc.)
2. Initialize the weights in the network. Although these are random values, the type distribution and the magnitude of the select values can impact how quickly and how well a network learns.
3. Employ one of many variants of gradient descent over some number of epochs (passes through the training set) until a set of weights are found that drive the loss function to a reasonable local minima.
4. Test the results on unseen data. If results are acceptable, stop. If results are not acceptable, either repeat steps 2 through 4 or consider running cross-validation to find a better value for one of the structural aspects (i, ii, or iii) of the network.

In spite of the complexity of working with NNs, a single-hidden-layer network was selected as the second to test LLM data augmentation. The number of input units are expected to range from roughly 5000 to 10000 based on the maximum number of tokens in the training and test sets.

## 5. Approach and Metrics

## 6. Summary and Conclusion

References


[0] reference related to the lack of labeled NLP data
[1] word embeddings reference
[2] transformer architecture reference
[3]
[4]
[5] https://github.com/MichaelSzczepaniak/WordEmbeddings/blob/master/WordEmbeddings.ipynb
[6] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4965448/
[7a] https://nlp.stanford.edu/projects/glove/
[7b] https://nlp.stanford.edu/pubs/glove.pdf
[8] https://en.wikipedia.org/wiki/Cosine_similarity
[9] https://www.baeldung.com/cs/cosine-similarity
[10] reference regarding the use of custom vs. canned embeddings
[a] Attention Is All You Need: https://arxiv.org/abs/1706.03762
[b] https://machinelearningmastery.com/transduction-in-machine-learning/
[c] reference related to the importance of prompt engineering
[d] https://www.kaggle.com/competitions/nlp-getting-started
[e] reference related to context window
[f] https://platform.openai.com/docs/models/gpt-3-5-turbo
[g] https://neptune.ai/blog/data-augmentation-nlp (uses same kaggle dataset)
[h] https://en.wikipedia.org/wiki/Prompt_engineering
[i] bag of words reference
[j] https://app.datacamp.com/learn/courses/chatgpt-prompt-engineering-for-developers
[k] https://www.promptingguide.ai/introduction/tips
[l] https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
[m] (from g) https://neptune.ai/blog/document-classification-small-datasets (uses same kaggle dataset)
[n] https://amitness.com/2020/02/back-translation-in-google-sheets/
[o] https://arxiv.org/abs/1901.11196 and https://github.com/jasonwei20/eda_nlp
[p1] https://github.com/albumentations-team/albumentations and
[p2] https://www.kaggle.com/code/shonenkov/nlp-albumentations
[q] EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks
https://arxiv.org/pdf/1901.11196.pdf
[r] A Survey of Data Augmentation Approaches for NLP
https://arxiv.org/pdf/2105.03075.pdf
[s] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing
[t] https://thenaturehero.com/chatgpt-vs-bert/
[u1] An Introduction to Statistical Learning, with application in R, Second Edition, page 137
[u2] An Introduction to Statistical Learning, with application in R, Second Edition, page 135
[u3] An Introduction to Statistical Learning, with application in R, Second Edition, page 335, 336
[u4] An Introduction to Statistical Learning, with application in R, Second Edition, page 340
[v1] DSCI 640 lecture slide 22
[v2] https://github.com/MichaelSzczepaniak/llmamd/blob/main/docs/lectures_dsci_640_1_thru_8.pdf
adapted from pages/slide 32
[v3] https://github.com/MichaelSzczepaniak/llmamd/blob/main/docs/lectures_dsci_640_1_thru_8.pdf
pages/slides 23-30
[v4] https://github.com/MichaelSzczepaniak/llmamd/blob/main/docs/lectures_dsci_640_1_thru_8.pdf
summarized from pages/slides 121-213

Data augmentation: A comprehensive survey of modern approaches (computer vision):
https://www.sciencedirect.com/science/article/pii/S2590005622000911