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

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 [1]. 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 [2]. 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. Word and Document Encoding

Before any model using text as input can be trained, that text must be converted to a vector. There are three main ways to do this vectorization: one-hot encoding, TF-IDF and word embeddings.

2.1 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 [3]. 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 [4]. 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 as shown in Table 1.

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

Table 1. Document for 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.

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

Figure 1. Simple BoW example

2.2 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 [7]. While this improves upon simple word counts, meaning of words and their relationships to each other are still not captured using this approach.

2.3 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 [8].

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:

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

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.

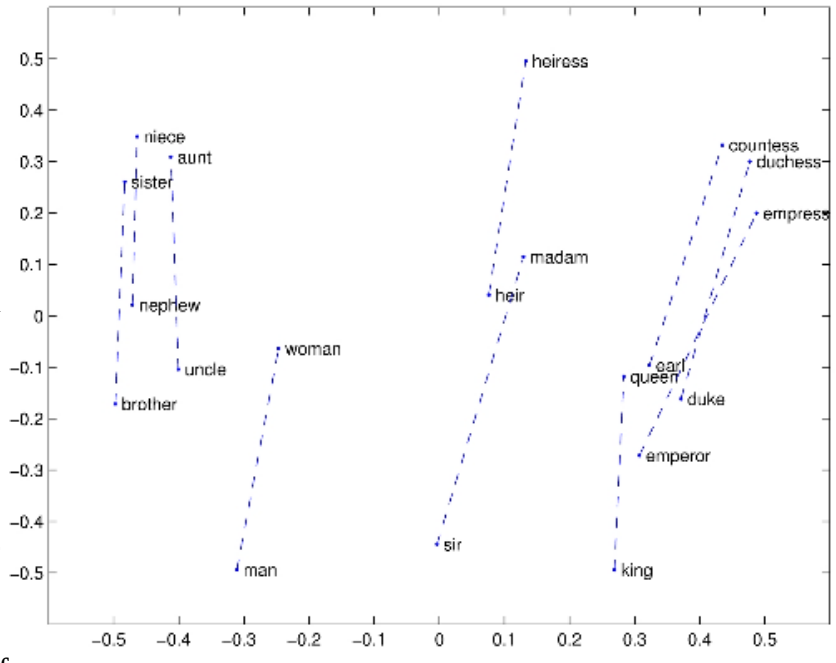

Figure 2. Word pairs with gender relationship

Figure 2. is taken from the GloVe website [8] 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 [10] 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. 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 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 [8]. Only one set of 300d pre-trained word2vec embeddings which were trained from a GoogleNews corpus are available [11]. 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.

There are a number of ways to encode documents when words are encoded as embeddings. One simple way is the calculate a mean of all the word embeddings in the document. Another way would be to compute a vector based on the minimum value in each embedding dimension, do the same for the maximum values in each dimension and then concatenate them together as described here [5]. Because this project is working with 140 max character tweets, each embedding word was simply concatenated with the next and padded with empty space to fill out a fixed-size representation for the tweet.

3. Data Augmentation Approaches

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 [12] 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 [1].

In the domain of NLP-related tasks, many data augmentation techniques have been developed [13]. Three techniques will be discussed here: 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 as shown in Figure 3. [14]. EDA is a set of simple and intuitive

transformations which can be applied to the original text data to generate augmented samples [15]. Using a tranformer leverages structural language knowledge encoded in an available LLM.

3.1 Back Translation

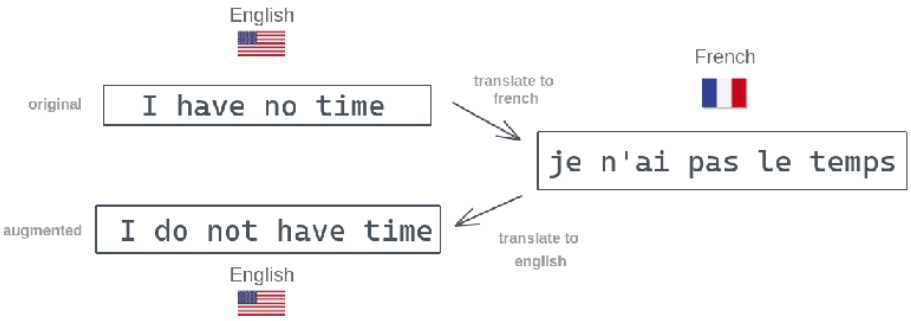

Figure 3. Example of Back Translation

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 [15]. 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 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 [16, 17]. 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 [18], 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.

3.3.1 Prompt Engineering

Once the decision has been made to utilize a generative AI model, designing the right prompts become the next order of business. Prompts are essentially the inputs supplied to the model. 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 [19]:

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 [20]:

- 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 saying what not to do – Focus on what the model is to do results in better results overall.

4. Text Classification Models

There are a number of choices for text classification models. Three types of 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 a 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. The β coefficients are typically determined by maximizing the likelihood function as described here [21 pages 134-137].

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. 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 [21, pages 339, 340].

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.

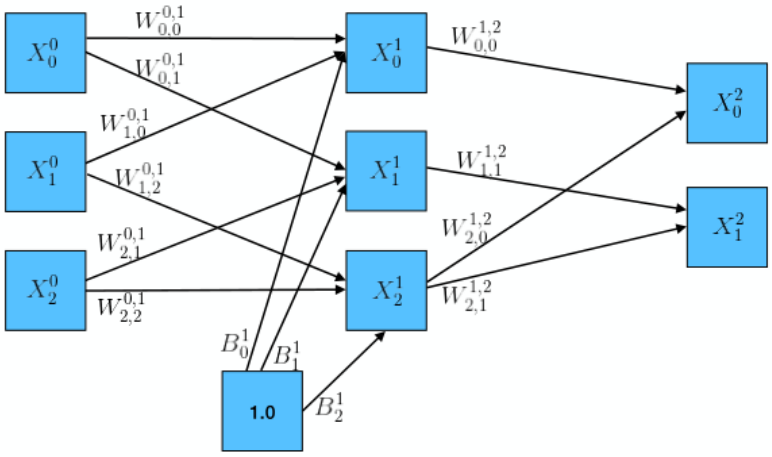

Figure 4. Neural network with single hidden layer
Biases (B^1_j) not added to input or output nodes.

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 4 [22, 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 ($X^{\text{layer}}_{\text{node}}$) multiplied by a weight associated with the connection ($W^{\text{from to layer}}_{\text{from to node}}$), adding a bias and then applying an activation function $g_{\text{layer}}()$ to that sum as shown in Figure 5 [22, slide 32].

4.3.1 Activation Function

There are many choices for activation functions, each having their own set of pros and cons as described here [22, slides 23-30]. The sigmoid and hyperbolic tangent (tanh) functions were historically the most frequently used, but had been displaced with ReLU and it variants in recent years as networks starting getting larger.

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

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

Figure 5. Calculating Figure 4. node outputs

Because we are using a single hidden-layer network, we won't need to worry about the "vanishing gradient" problem [22, slide 14], so a classic sigmoid activation function should be fine for the hidden

layer. For the output layer, since we are doing binary classification, using a sigmoid for the output layer should also be a good choice [22, slide 26] and allows us to keep things simple.

4.3.2 Number of hidden layers

After selecting an activation function for hidden and output layers, additional design choices must be made before an NN-based model can be implemented. Other choices include:

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

A single hidden-layer NN with a large number of hidden units has the ability to approximate most functions [21, page 407], however many complex NN architectures have evolved to better address specific types of problems. Because we don't require state-of-the-art performance on the text classification task, a single hidden-layer NN was selected to evaluate the impact of LLM data augmentation. To keep things simple, the hidden layer will be fully connected to both the input and output layers.

4.3.3 Nodes in each layer

After deciding on the number of hidden layers, the number of nodes or units in each of these layers must 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, it will need to be determined through experimentation. To keep things simple, each layer in the network will be fully connected.

4.3.4 Solving for the weights

Once the network structure has been fully specified, the weights and biases are typically learned in a supervised fashion using a gradient descent algorithm. The work flow generally looks like the following:

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. Compute the gradients for each weight with respect to the loss function and update the weights for each training sample. Continue computing gradients and updating weights 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 listed in i, ii, or iii in the previous paragraph.

5. Metrics

Because accuracy is one the most intuitive metrics for measuring model performance, it will used in this project. However, one model can be noticeably better than another model while having the same accuracy. This happens when one model consistently assigns a higher probability to the correct class and lower probability to the incorrect class as demonstrated in this example [23]. To more robustly account for how well models assign class probabilities, ROC curves and the area under these curves (AUC) will be used to evaluate model performance under the two training scenarios. Building precision-recall (PR) curves was also considered, but because our classes are roughly balanced, it was not considered necessary [24].

6. Summary and Conclusion

To evaluate the impact of NLP data augmentation, the following components needed to be selected:

1. Word and document encoding approach
2. Data augmentation approach
3. Text classification models
4. Metrics by which models are to be evaluated

Word and document encoding approach - Twitter GloVe embeddings were selected as the word-level encoding scheme over one-hot encoding, TF-IDF and other embedding options. Having a pre-trained Twitter embedding available provided the benefit of a custom embedding without having to build it from scratch. Because the data being analyzed is subject to the old Twitter character limit (140), concatenating the word-level embeddings was chosen as the document-level encoding scheme because it retains positional and semantic information while keeping the vector size reasonable.

Data augmentation approach - The ChatGPT 3.5 turbo transformer was selected to generate augmented samples over the BERT transformer, the Back-Translation (BT) approach and the Easy Data Augmentation (EDA) approach. While BERT is potentially more powerful than ChatGPT 3.5 turbo, it's configuration overhead was considered a significant drawback. The Back-Translation approach required a translation engine (e.g. Google Translate) and generated samples that were not significantly different from the originals. The EDA approach was doing something similar to what transformers were doing, but were not taking advantage of the knowledge of language structure.

Text classification models - A logistic regression model and a single hidden-layer neural network were selected as the two models used to evaluate the impact of data augmentation. Tree-based methods aren't typically used for text classification. Logistic regression has no hyper-parameters to determine and provides a good baseline from which to compare the impact of training data. A single hidden-layer neural network is complex enough to approximate most functions and typically does better with more training data so it should pick up an augmented data if that data is adding information to the system.

Metrics by which models are to be evaluated - Accuracy is the most intuitive metric used for classification and was selected as one of the metrics. Accuracy can be misleading because it does not account for the extent to which probabilities are assigned to the correct and incorrect classes. To account for shortfall, ROC curves and the accompanying area under the curve (AUC) were selected as an additional metric. Precision-recall curves were also considered as a metric, but because the classes were relatively balance, it was not considered necessary as part of model evaluation.

References

[1] 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
[2] *Natural language processing with disaster Tweets*. (n.d.). Kaggle.com. Retrieved March 18, 2024, from https://www.kaggle.com/competitions/nlp-getting-started
[3] (N.d.-c). Machinelearningmastery.com. Retrieved March 18, 2024, from https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/
[4] (N.d.-d). Machinelearningmastery.com. Retrieved March 18, 2024, from https://machinelearningmastery.com/gentle-introduction-bag-words-model/
[5] Szczepaniak, M. (n.d.). *Word Embeddings*. Retrieved March 18, 2024, from https://github.com/MichaelSzczepaniak/WordEmbeddings/blob/master/WordEmbeddings.ipynb
[6] 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
[7] (N.d.). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/
[8] Pennington, J. (n.d.). *GloVe: Global Vectors for word representation*. Stanford.edu. Retrieved March 19, 2024, from https://nlp.stanford.edu/projects/glove/
[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] (N.d.-b). Baeldung.com. Retrieved March 19, 2024, from https://www.baeldung.com/cs/cosine-similarity
[11] *GoogleNews-vectors-negative300.bin.gz*. (n.d.). Google Docs. Retrieved March 19, 2024, from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing
[12] (N.d.-c). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
[13] 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
[14] 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/
[15] 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
[16] (N.d.-d). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/transduction-in-machine-learning/
[17] 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
[18] Budhathoki, S. (2023, March 9). *ChatGPT vs. BERT: Battle of the transformer*. The Nature Hero. https://thenaturehero.com/chatgpt-vs-bert/
[19] (N.d.-e). Datacamp.com. Retrieved March 19, 2024, from https://app.datacamp.com/learn/courses/chatgpt-prompt-engineering-for-developers
[20] *General tips for designing prompts*. (n.d.). Promptingguide.Ai. Retrieved March 19, 2024, from https://www.promptingguide.ai/introduction/tips
[21] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2022). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer. Chapter 4.3
[22] 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 http://docs/lectures_dsci_640_1_thru_8.pdf
[23] *RPubs - assessing goodness of classification models*. (n.d.). Rpubs.com. Retrieved March 19, 2024, from https://rpubs.com/mszczepaniak/classificationgoodness
[24] (N.d.-g). Machinelearningmastery.com. Retrieved March 19, 2024, from https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/