Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Chair for Mathematical Information Science
Prof. Dr. H. Bölcskei

Semester Thesis in Information Technology and Electrical Engineering

Spring Semester 2021

Siddhant Ray

Attentive Neural Networks for News Classification

Supervisor: Dmytro Perekrestenko

May 2021

Abstract

We present a model for a downstream news classification task on a multi-category news dataset. We develop the model using a transformer based neural network architecture, which we use to classify the news items into its category labelled in the dataset. Furthermore, we propose a new statistical algorithm, which helps learn the degree of overlap between similar news categories in the dataset, using representations from our model. Finally , we fine-tune our dataset based on our algorithm, and re-run our model on it, showing significant improvement in performance over the the run carried out on the original dataset.

Acknowledgments

I would like to thank Prof. Dr H. Bölcskei for providing me the opportunity to carry out this project in his research group. I would also like to thank my supervisor Dmytro Perekrestenko for his constant help and guidance for the entire duration of the project.

Contents

Chapter 1

Introduction

1.1 Motivation

The Internet today is a vast source of news to its users, providing countless websites and pages which provide latest news to the viewers. With the advent of the amount of news and news related data on the internet, it becomes an interesting problem to view the problem of fast, easy and efficient access to the desired type of news for a particular user. In this light, we study the possibility of building a automated news classification model based on a neural network learning method. The utility of such a news classification model can be multi-fold. With a robust news classification model, it can be possible to build custom applications which help users in faster information retrieval for a news category query, detection of possible fake news from genuine news, improve existing misclassified datasets, understand contextual similarity between news items, sentiment analysis of text and so much more. In this project, we attempt to build a robust news classification model on a multi-class news dataset, which tries to improve over existing techniques. We also try to build a new algorithm which can learn similarities between news categories using our training model. We will introduce specifics of the dataset and model in a later chapters dedicated to them. Our implementation is fully open source and the complete code can be found below. [1]

[1] https://github.com/Siddhant-Ray/NewsClassifier

1.2 Related work

News classification is a special area of application for the problem of text classification in the world of Natural Language Processing (NLP). Over time, several learning models have been explored in the field of improving text classification models. For any text classification problem, the primary task becomes transforming pieces of raw text to numerical vectors, usually known as word embeddings. One of the oldest known word embedding techniques is the TF-IDF vectorization technique [1], where words were given numerical importance based on its frequency of appearance in both the page, and the entire document.

Then, with the evolution of pre-trained models for transfer learning, where a neural network model trained on a huge text datasets (known as corpuses) could be initialised with the pre-learnt weights, for new downstream tasks the user needed. Few pre-trained word embedding models which came up were word2Vec [2] and GloVE [3], which were able to learn word embeddings by mapping a relationship to the surrounding words, but had the same embedding for a word despite the context being different. These models learnt embeddings for an entire word at a time. Then, with the introduction of FastText [4], it was shown that words can be split instead as character *n-grams* and learning *n-grams* helps learn the embeddings for rarer words, which appear infrequently the corpus, much better. We finally have BERT [5], which is the state-of-art currently in learning word embeddings, which learns the embeddings of the words based on both the left and right neighbourhood context of the given word.

Recently, techniques are being explored to learn the embeddings for entire sentences over words, where the model can then be used to compare similarity in entire paragraphs, generate entire volumes of text etc. SentenceBERT [6] and Universal Sentence Encoder [7] show instances of the current state-of-art in the sphere of sentence embeddings.

The problem of building a new classification model using machine learning techniques is also not new. Attempts are constantly made in this area, to improve existing models from time to time, in order to make them more robust for further downstream applications. The article in [8] shows a news classification model built using an ensemble of Recurrent Neural Network techniques. We build our model using a newer version of the dataset used by them, and attempt to improve upon the performance of the model. However, in their model, they learn the word embeddings directly from the news dataset. We do not do that and we will explain our ideas and approach in the subsequent chapters. Similarly, other news classification models exist and a lot of research has been in done in the area.

In [9], they show an approach to classify the dataset based on generation of pseudo news items for data augmentation. However, we felt it would be a good approach to make the model robust without relying on the need for generated text, and hence we do not use that approach either in our model.

Finally, we present in our work, a way to learn overlap between multiple classes in our news classification problem. The sphere of using neural network based models to learn class overlap and hierarchy in the dataset is relatively new. In all, some research for learning class hierarchies has been carried out in the field of computer vision, for ranking images.In [10], they show one such research area, where images were ranked maintaining the hierarchy of words in the WordNet database. However, we follow a different approach in our project for learning class overlaps using a statistical method, which we will introduce in a later chapter

Chapter 2

Dataset

2.1 Source and description

The dataset for our model has been taken from an open source dataset available on Kaggle. [11] The dataset contains $\sim 200k$ news descriptions obtained from HuffPost [12] over the years of 2012 to 2018. Our version of the dataset is a classified dataset with 41 classes of news labels. Each news description contains several fields such as "category", "headline", "author", "date", "description" etc. For our project, we use the fields of "headline" which contains the main summary of the news item and "description" which contains a short description of the news item, as our feature for training the model.

2.2 Data preprocessing

Our primary evaluation of the dataset returned some key observations which would be useful for cleaning up our dataset. For our model, we combine the headline and description every news item, into a single phrase , which we use as our training samples. We will refer to these samples as news entries subsequently. Then, to begin with, we measure and plot the number of news entries per class of news for our project.

From Figure 2.1, we can see that there is a huge imbalance in the number of news entries per class. For example, *Politics* has $\sim 30k$ news entries whereas some other classes like *Women* and *Impact* have only around $\sim 4k$ news entries. We take note of this excess class imbalance in our dataset and will incorporate methods to mitigate its effect when we train our model, else we realise it will be harder to draw accurate inferences.

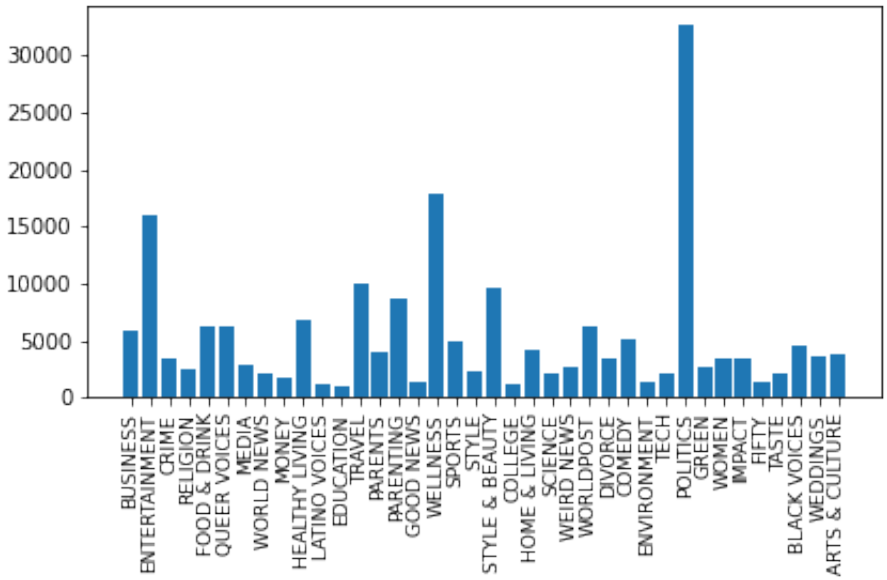


Figure 2.1: Number of news entries per category

Another metric we take into account is the average number of words (or average length of an entry) per news entry for every category. This gives us an indication of the possible imbalance in the length of news entries in a given class, which will be useful for us when our model tries to learn the context of the words in a given news entry.

From Figure 2.2, we see that the distribution of the average number of words per category of category is far more uniform. This means that every category of news has a good length of its description , which makes the entries in every category of news representative for learning. For our dataset, we have a mean length of news entry as 27.34 with a standard deviation of 5.05. Based on these results, we safely conclude that the news entries is every class are representative for learning. If the descriptions in a particular class were too short, it would hinder the model from learning in the training process, and we need to incorporate a method to handle that.

Based on our analysis of the description lengths of the news entries, we decided to only keep news entries which have an overall length of ≥ 5 as entries of lower length did not seem meaningful nor representative. For example, a news entry from the category *Arts* had the overall description as just *"Another Hero Departed."* with length 3 which is neither representative of the category nor meaningful. In comparison, we also present a

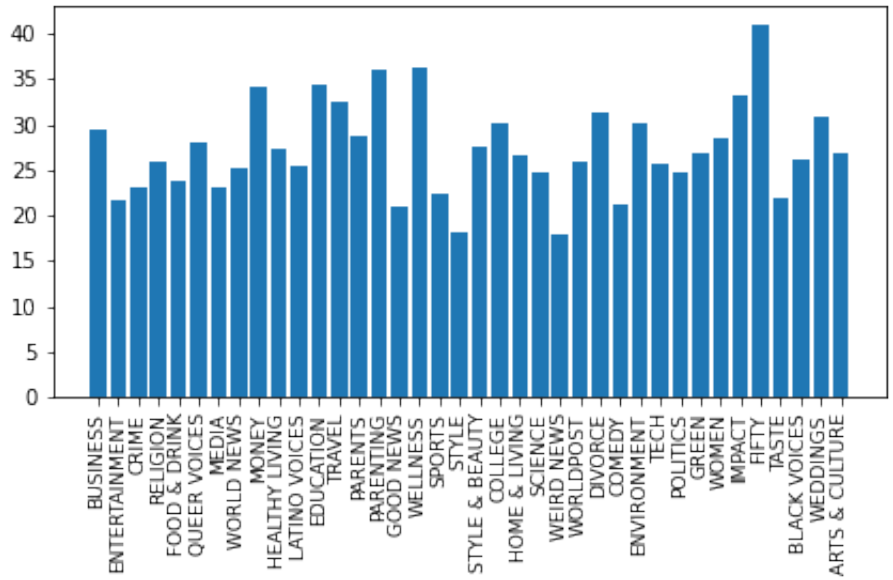


Figure 2.2: Average number of words per news entry in a category

sample from the category of *Politics* below :

Trump Lawyer Attended DOJ Meeting On Confidential FBI Informant. Emmet Flood, the White House attorney dealing with the Russia probe, was present at a controversial DOJ meeting about the investigation.

Finally, based on our analysis of the entries in the combine the classes of *The World Post* and *World Post* into a single class of *World Post* and the classes of *Culture and Arts*, *Arts and Culture* and *Arts* into a single class of *Arts and Culture*. We attribute this to classification in the original dataset and hence, combine them into single categories.

With this, we are left with 38 categories of news classes and a total of 198123 news entries which we will use for our training and validation process in the subsequent chapters.

Chapter 3

Background and Preliminary Work

3.1 Recurrent Neural Networks

Recurrent neural networks (RNN) have been the state-of-art model for sequence classification tasks for a long time. For news classification, it is no different as the embedding vectors generated for the individual words of any news entry, can be processed as a sequence and thus fed to a recurrent neural network model to generate fixed size output vectors for a given language model input. We studied a RNN based model from Facebook AI Research : Infersent, [13] where Alexis C. et al showed a sentence vectorization which could be used for our downstream news classification task. In this model, they learn the word embeddings on the given dataset using a pre-trained embedding model called FastText, where an $n - gram$ character level model is used to learn the word embeddings, instead of earlier whole word embedding techniques. The advantage of this $n - gram$ model is that it helps capture the embeddings for rarer words much better as greater context combinations are created by generating several character $n - grams$ and learning the embeddings for them. Infersent uses these generated word embeddings and passes them through a bi-directional LSTM recurrent neural network, which generates fixed size vectors for entire sentences and phrases. This made sense for our downstream task as we pass our news entries through a neural network architecture like Infersent, which gave us fixed size representations for all our dataset news entries.

For our first attempt, we tried a model for training in which we learnt the embeddings for our news entries using Infersent's model and then, we

used a self-normalizing feed forward neural network [14] as a classifier to predict the news categories for our descriptions. Our overall pipeline for this architecture was as follows in Figure 3.1 :

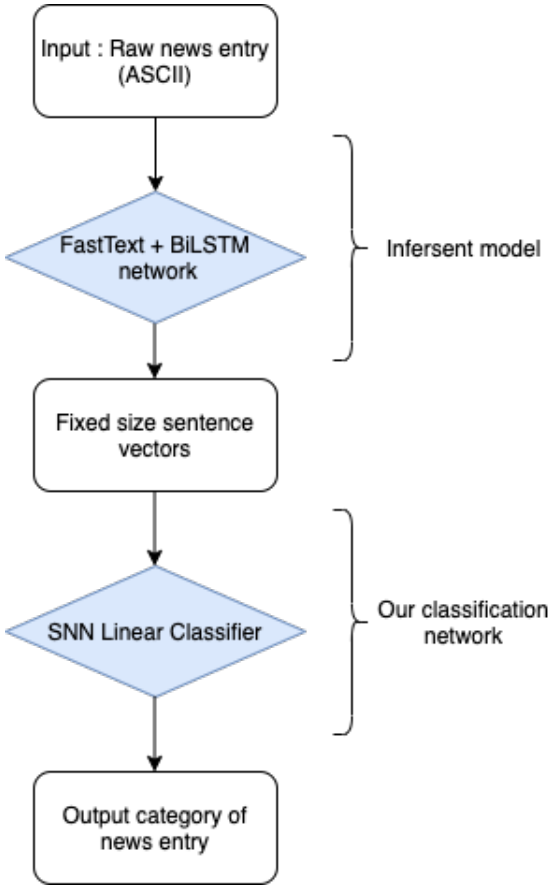


Figure 3.1: RNN based news classifier

This RNN based classifier did not perform as well as expected and we identified the following reasons for the same :

- In our Infersent model, the embeddings were learnt purely based on the pre-trained FastText word vectors and there wasn't any non-trivial way to fine-tune the same for our dataset, hence the representation for the embeddings were not the best for our classification task.
- A fundamental limitation for traditional RNNs is that the input sequences must be processed in order. This limits the flexibility during the training process, making the process slower and allows the user to leverage less customization for the same.

- Finally, we explored the possibility of incorporating an attention mechanism [15] into the RNN network, which would let us focus on particular portions of the sequences by using calculated attention weights. However, for this too, there didn't exist a non-trivial, scalable method and we hence, decided to shift our focus to an alterative architecture for our project.

3.2 Transformers : Leading to BERT

The state-of-art models which exist today for processing sequential data, especially for language modelling tasks such as classification, text generation, translation etc. are based on transformer architectures.

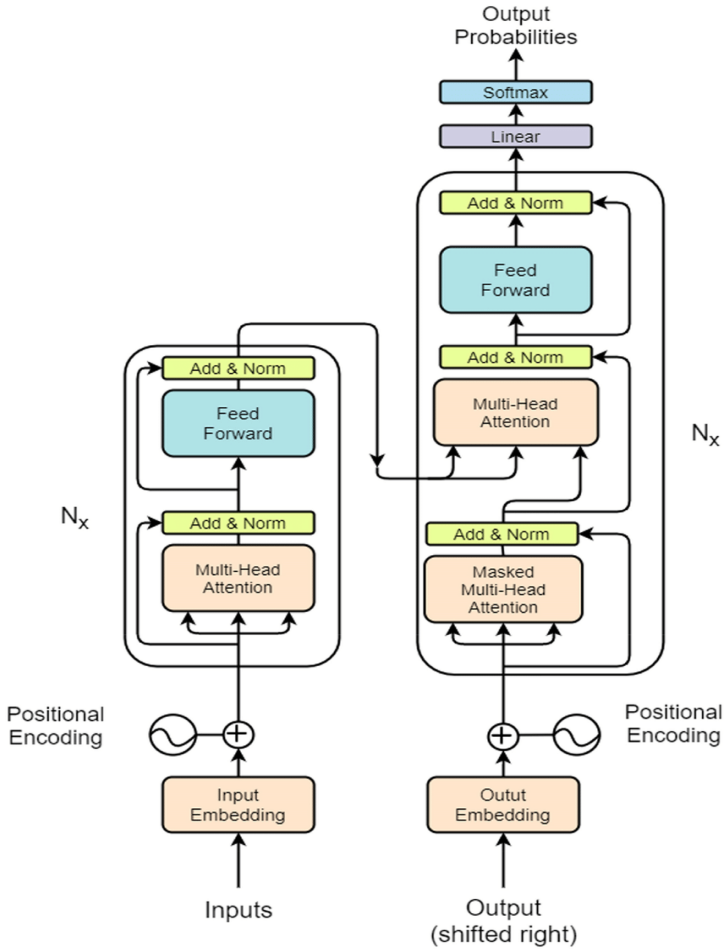


Figure 3.2: Transformer Architecture, Source : Original Paper [15]

Figure 3.2 shows the original architecture of the transformer model. Transformer architectures offer significant improvement over traditional RNNs as they do not require the input tokens to be processed in sequential order. They offer higher parallelization in the training process as they process all tokens together and use attention weights to ensure focusing on appropriate parts of the sequence data. This is done using the attention mechanism within the encoder and decoder. In the original transformer model, the attention head allowed the model to focus on the relevant part of the sequence, using an attention-weight matrix. The transformer uses self-attention inside the encoder on the input, and cross-attention on the output of the encoder which goes to the decoder. The multi-head attention allows for multiple sets of attention weights to be used within the single encoder or decoder stack. Hence unlike RNNs, it is no longer needed to process the input sequence in order. Hence, transformers have shown significant utility in language processing tasks, where it can process input sequence data in parallel, making it faster and modular to train on larger corpuses of data, which creates better pre-trained models.

A transformer model used in Natural Language Processing (NLP) tasks today is Bidirectional Encoder Representations from Transformers or BERT. The BERT model is extremely useful for downstream NLP tasks as it captures both the left context and right context of a given word in a sequence, and produces a contextual word vector. This is an improvement over older word vectorization techniques such as word2vec, as more information can now be captured about the context of the word, which makes the word vector more representative. The BERT model preserves only the encoder stack from the original transformer model, and uses a self attention mechanism on its encoder layers. The original BERT model was trained with 12 encoder layers with 12 bidirectional self-attention heads, followed by a feed forward network of dimension 768. The model is pre-trained from unlabeled data (unsupervised training) extracted from the BooksCorpus with 800 million words and the English Wikipedia with $2,500$ million words. Overall, the BERT base model has 110 million parameters to be learnt.

Chapter 4

Methodology

4.1 Neural Network Classification Model

Based on our studies, we decided to use a BERT based architecture for our classification task. BERT provides an extremely robust pre-trained model for any downstream NLP task, hence our architecture consists of a pre-trained BERT model, followed by a classification head which consists of linear layers and dropout layers. However, we make a small design decision here, instead of using the BERT base pre-trained model, we choose to use the DistilBERT pre-trained model [16]. The reason for this is that DistilBERT is a much faster and smaller implementation of BERT, while preserving the performance of the original BERT base model. In their implementation, Victor et al show that DistilBERT reduces the size of the BERT base model by 40%, is 60% faster, while retaining 97% of the language understanding and performance of the original BERT base model. Due to limited hardware resource we had for our training process, we decided to use the DistilBERT pre-trained model for its speed, size and efficiency. In our implementation, our DistilBERT model has 6 hidden layers in the encoder stack, 12 attention heads in each attention layer and in all 66 million parameters. The overall architecture for our model is as presented below :

For our model in Figure 4.1, we first use a DistilBERT tokenizer to generate a numerical representation of the sequence that will be fed to the DistilBERT model, which are in the form of *input_ids*. For this, we pass our raw input news entries to the tokenizer. Since our DistilBERT model needs to be fed with fixed size tokenized inputs, we pad the tokenizer outputs to fixed sizes by adding empty tokens. It is also necessary to generate an attention mask for each tokenized news entry, which decides which token to focus on, and which to ignore. The *attention mask* should prevent

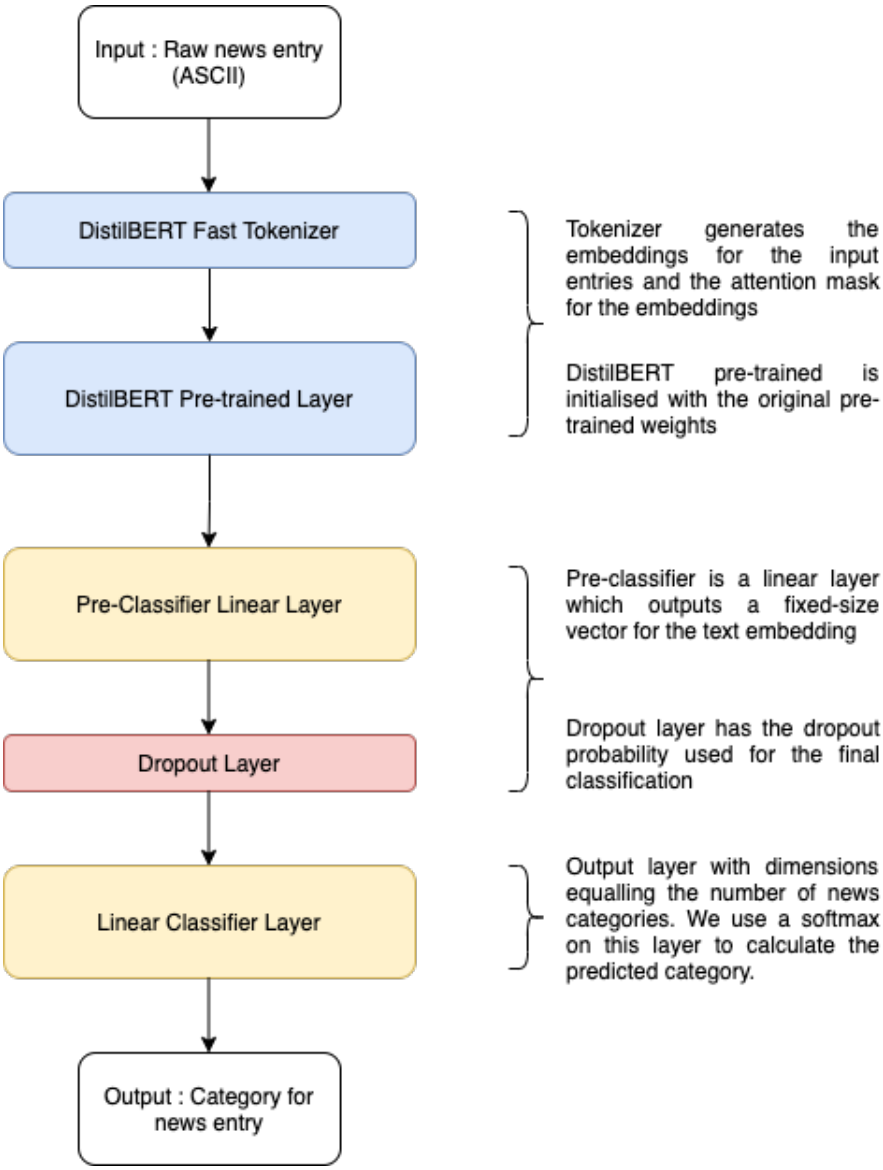


Figure 4.1: News Classification Model Architecture

performing attention on the padding token indices and the mask takes a binary value of 0 to represent tokens that are masked and 1 to represent tokens which are not masked. The base implementation of our model is from Huggingface's implementation of transformer models [17], on which we add our modifications.

We design our model to pass our tokenized input into our model, which consists of a pre-trained DistilBERT model, to which we attached a multi-stage linear classifier. Our classifier consists of a pre-classification linear layer, which pools the output of the DistilBERT model into a fixed size dense vector (we kept the dimension for this pre-classifier as 786, same as the dimension of DistilBERT's dimension, similar to the original implementation provided by the model's authors). We added a dropout layer after this pre-classifier linear layer to prevent overfitting and finally a fully connected linear layer which acts as a classifier, with dimensions equalling the number of our news categories. Before passing the output to the dropout layer, we also add a ReLU activation on the intermediate output (not shown in Figure 4.1)

Our linear classifier head produces as an output, a series of logits for every category, on which we use a softmax (not shown in the Figure 4.1) function to calculate the output probability of the news entry belonging to the particular class. We mention more details about the output, loss function and its understanding in the further chapters.

Chapter 5

Training

For our training, we go back to our dataset and divide into a training and validation set in a $80\% - 20\%$ ratio respectively. This split is done using a random selection of news entries from our overall dataset. Based on the number of entries retained in our dataset after pre-processing, we have around $\sim 158k$ samples in our training set and $\sim 40k$ samples in our validation set. One individual training sample for us, consists of an input news entry and its actual category from the original dataset, which we will refer to as the *target class* for our project. We tokenize the input news entry using our tokenizer and make an input training pair consisting of the tokenized input and the *target class*. We use the PyTorch [18] library and its methods to implement our model, training and validation functions.

We train our network by using batches of inputs, with a batch size of 16. This size is chosen in accordance with the limits of the hardware resources on which we train. For training, we use 12 GB Nvidia Graphics Processing Units (GPU)s, accessed via the CUDA API. We run our training process for 9 epochs of training, validating our results after every epoch on the validation set. The entire training and validation process for our model takes around 7 hours on a single GPU.

For our loss function, we use CrossEntropyLoss (CELoss) calculated between the output of our classifier's final layer and the *target class*. This single metric of CELoss, combines both a LogSoftmax function and a Negative Log Likelihood (NLL) loss function, into a single measure. This helps us as we no longer need to explicitly calculate the Softmax function on the output logits of our linear classification output. Mathematically, the CELoss function can be expressed for item x and class c as:

$$\text{loss}(x, c) = -\log\left(\frac{\exp(x[c])}{\sum_j \exp(x[j])}\right)$$

or

$$\text{loss}(x, c) = -x[c] + \log \sum_j \exp(x[j])$$

However, we had a problem of excess class imbalance in our dataset, and to mitigate this, we decided to use a weighted CELoss, in order for a fairer learning process. We set the weights of every class as

$$w[c] = \frac{1}{\textit{\# of items in c}}$$

The weighted CELoss function becomes :

$$\text{loss}(x, c) = w[c](-x[c] + \log \sum_j \exp(x[j]))$$

As we train in mini-batches of size 16, the losses are weighted averaged for every mini-batch. Hence, for our final loss function we get :

$$\text{loss} = \frac{\sum_{i=1}^{N} \text{loss}(i, c[i])}{\sum_{i=1}^{N} w(c[i])}$$

For our optimizer, we use the Adam optimizer [19]. As our model consists of fine-tuning the pre-trained DistilBERT model on a supervised downstream classification task, we had to choose a low learning rate, to prevent the model from converging too fast. We train our model with a learning rate of $0.00001 = 1e - 5$. Finally, we also add a regularization parameter to prevent overfitting during our training process, in the form of weight decay. We set this as 0.0001 for *bias* and *LayerNorm.weight* and 0.01 for other parameters. We also set a dropout probability of 0.2 for our dropout layer. A summary of our hyper-parameters for training is presented in Table 1 .

Table 1: Hyper-parameters for our network

Hyper-Parameter	**Value**
Number of epochs	9
Learning rate	1e-5
Optimizer	Adam
Batch size	16
Loss function	Weighted CELoss
Dropout probability	0.2
Weight-Decay	0.0001 for Bias, LayerNorm weight and 0.01 for rest

Chapter 6

Evaluation

6.1 Initial performance evaluation

For our evaluation, we use the validation set of our dataset after the training process is completed. For our preliminary evaluation, we consider the metrics of accuracy for the top predicted class and also for the top three predicted classes. The accuracy for the validation set of our model is calculated as :

$$\text{accuracy top1} = \frac{\text{\# of (predicted class} == \text{target class)}}{\text{total \# of news entries}}$$

$$\text{accuracy top3} = \frac{\text{\# of(any top3 predicted classes == target class)}}{\text{total \# of news entries}}$$

However, as it is a multi-class classification problem, the accuracy cannot give us the complete information required which shows the performance of our model. We also compute the F1 scores for every news category using our validation set in order to check which categories perform the best and which categories are confused the most with each other. The F1 score is interpreted as the weighted average of precision and recall, and it depicts how well a particular category of news performs for us. As the raw F1 score for each category is hard to interpret, we calculate the unweighted mean F1 score over all categories for reporting. We calculate the F1 score $F1$ for every category defined as :

$$F1 = 2 * \frac{P * R}{P + R},$$

where P is the precision given as

$$P = \frac{TP}{TP + FP}$$

and R is the recall given as

$$R = \frac{TP}{TP + FN},$$

where TP is the number of true positives, FP is the number of false positives and FN is the number of false negatives.

Finally, as it is a multi-class classification problem for us and we use the top three predicted accuracy as a metric, we also calculate the Mean Reciprocal Rank (MRR) for our model. The MRR is an information retrieval measure which computes the fraction of each top correct response guessed compared to the actual response by assigning it a rank. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for first place, $\frac{1}{2}$ for second place, $\frac{1}{3}$ for third place and so on. The value of the MRR lies between 0 and 1. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q and is given as :

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_{\mathrm{i}}}$$

For our model, we calculated the values as follows after 9 epochs of training in Table 2:

Table 2: Evaluating the base model

Performance Metric	**Value**
Accuracy (Top prediction)	65.67%
Accuracy (Top 3 predictions)	87.75%
Mean F1 Score	0.5920
MRR	0.7574

We also plot a confusion matrix heatmap for all our predicted news categories for the validation set vs our actual news categories. Figure 6.1 shows the plot for the same. For our code for our performance metrics, we have used reference implementations from Scikit-learn [20].

Our base model does not perform extremely well but it is promising for complex classification task with these many categories (38) as ours. We see that the top3 accuracy is a good metric for our problem as several items in our dataset are subjective to classification in more than one category.

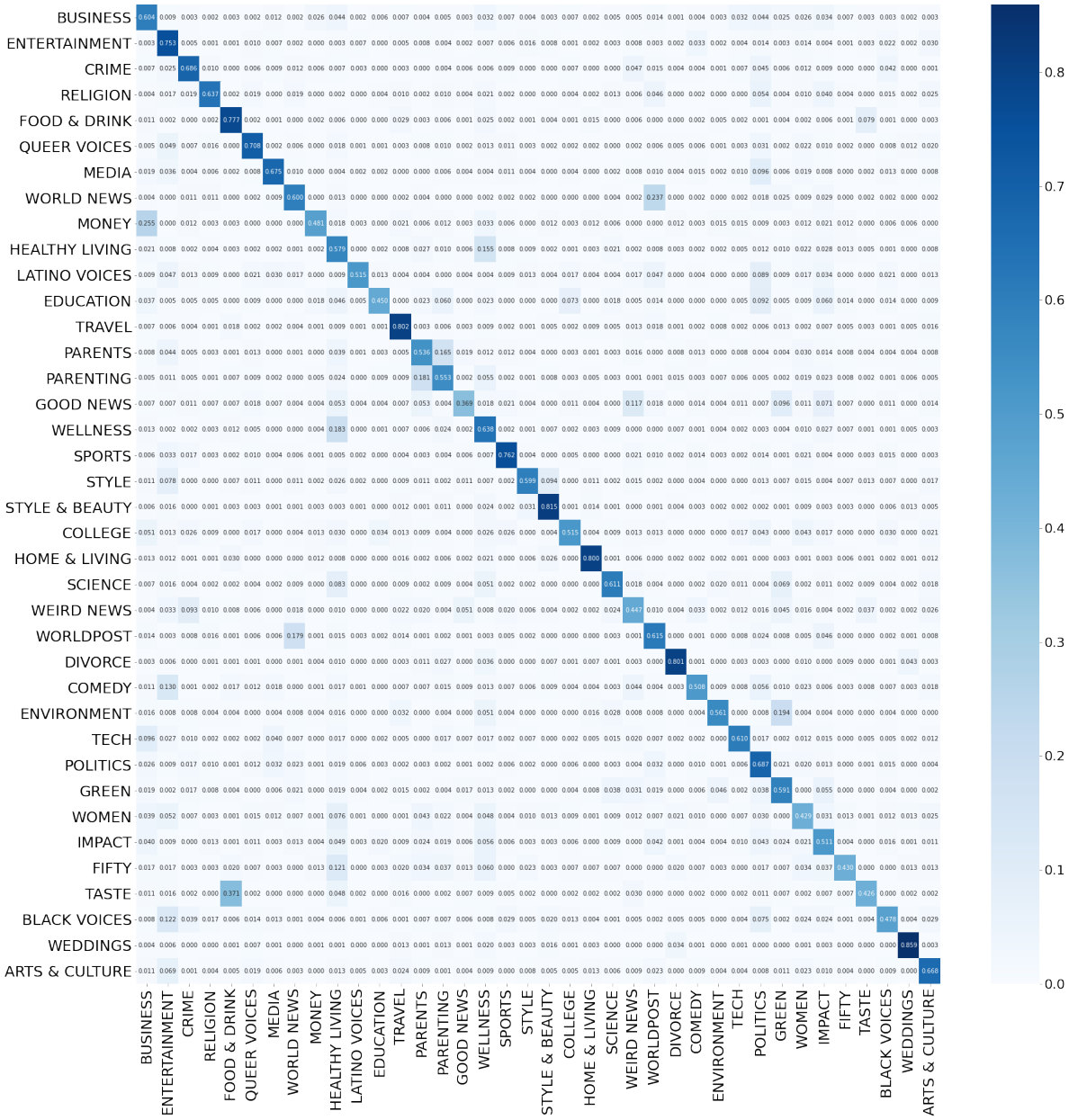


Figure 6.1: Confusion Matrix for all news categories

For example, it might be possible for a news entry to be classified in both "BUSINESS" and "MONEY" or both in "EDUCATION" and "COLLEGE". We also see that some classes perform extremely well in Figure 6.1.

Classes like "HOME & LIVING", "WEDDINGS" , "TRAVEL" perform quite well and most of the items are classified correctly (we use high F1 score as a metric to measure this). However, many classes such "COMEDY", "WOMEN", "MONEY", "HEALTHY LIVING", "IMPACT" have very low F1 scores, and are easily confused with other classes, often with particular other ones. In the next section, we explore a method to mitigate this problem and improve the performance of our model.

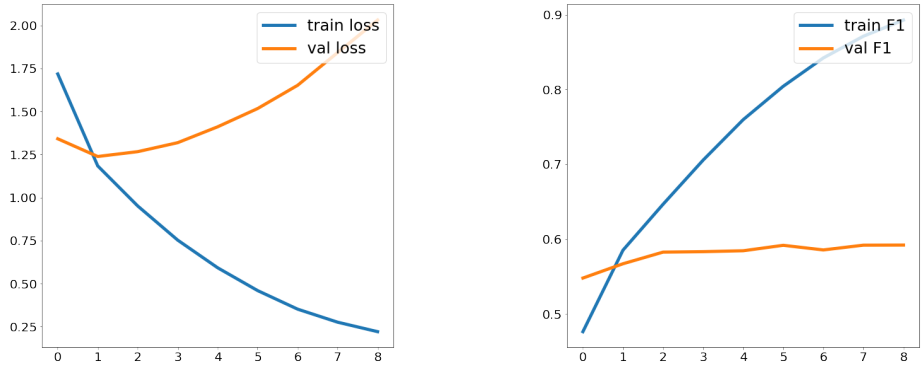
6.2 Analysis for fine-tuning the performance

We identified the following fundamental problems in our dataset which causes the model to perform below its expectation.

- There is a huge class imbalance in our dataset like mentioned before. Classes like "POLITICS" have almost 10 times the number of news entries as compared to a class like "WOMEN". Though we used a weighted loss function while training our model, we feel it might not have been enough to mitigate the problem completely.

- On re-analysing the dataset, we also identify the possibility of overlapping classes in our dataset, which have very similar news entries in multiple categories. For example the news entry *"Quarter Of World's Land Will Be Permanently Drier If Paris Climate Goals Not Met. Countries need to work to prevent the Earth's temperature from rising more than 1.5 degrees."* is classified as "GREEN" in the dataset, but may also very well come under "ENVIRONMENT". Similarly, *"Dunkin' Donuts Coffee Is Being Turned Into A Stout Beer. Dark Roasted Brew is the first beer to be made with the company's dark roast beans."* is classified under "TASTE" but may also be classified under "FOOD & DRINK". Our confusion matrix in Figure 6.1 indeed shows mis-classification between these pairs.

We attribute this problem to the dataset having some categories which are very similar to each other, which makes our model classify news entries from those classes into similar classes. This happens as our model is a BERT based architecture, which learns contextual word embeddings in a given sequence. For very similar sequences, it is possible that our model



(a) Model loss

(b) Model F1 score

Figure 6.2: Model loss and F1 curves

can misclassify the items into incorrect classes, if the classes overlap in context.

Thus, we attempt to build an algorithm which can give us a justified indication of which news categories are extremely similar and should possibly be combined to form a single category. Also, we attempt to make the algorithm return an indication of a class which overlaps with multiple classes, making it very widespread and hard to classify items into. Such a news category can possibly be removed from the dataset. This combined approach will both reduce overlapping classes and solve the class imbalance problem to an extent, as combining classes will lead to higher news entries in the new class.

To visualize the similarity of the embeddings of our news entries across different categories, we extract the output of the pre-classifier layer of our network before the final classifier layer maps it to the output category. As the dimension of output of this layer is high i.e. 786 for every embedding, we use t-distributed Stochastic Neighbor Embedding (t-SNE) [21] to reduce the dimension of the embedding vector to 2, in order for easy visualization. t-SNE is a useful metric as it preserves maximum information of the vector while projecting it in a lower dimensional subspace by preserving local distances between the original vectors.

One thing we had to ensure is to understand the similarities between the categories using t-SNE on embeddings from our trained model is that the model should not over fit on the data. If this happens, we may fail to get an accurate representation of class overlap as the model may forcibly

separate the categories which in reality overlap a lot. To ensure this, we opt for early stopping in our training phase, before the model starts to heavily over fit.

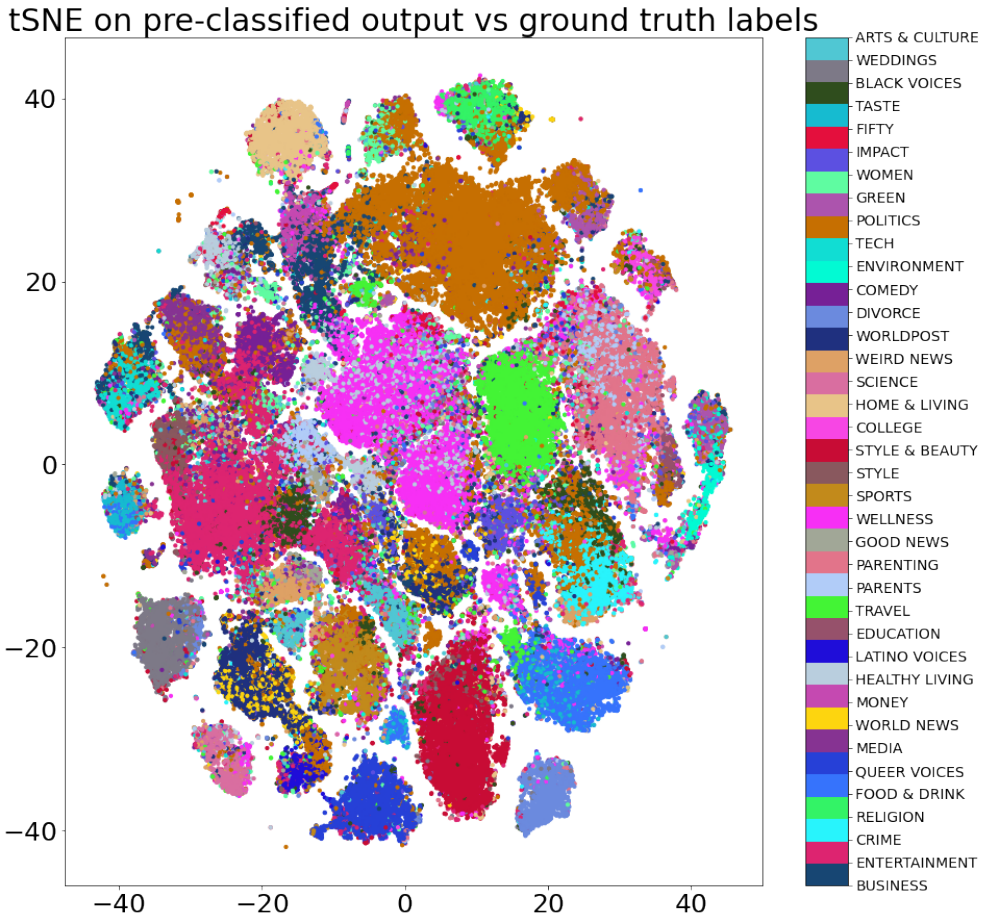
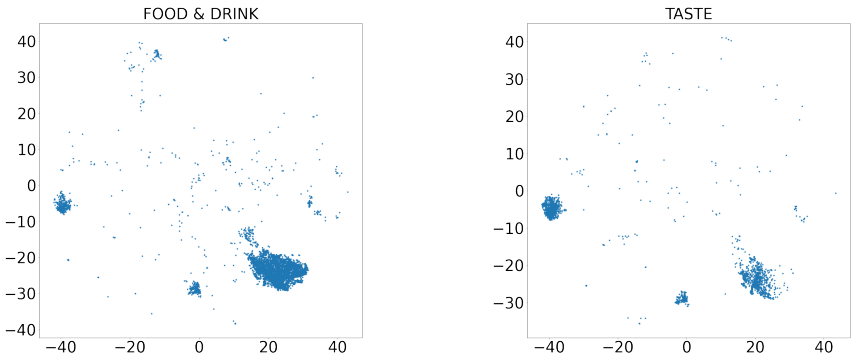


Figure 6.3: t-SNE visualization for pre-classified embeddings

As we see in Figure 6.2, our model starts to overfit extremely fast despite our regularization and dropout measures. However, as it is a multi-class classification problem for us, we feel the F1 score is a better metric for evaluation, as we see that after epoch 2, our F1 score doesn't improve much further and gap between the train F1 score and the validation F1

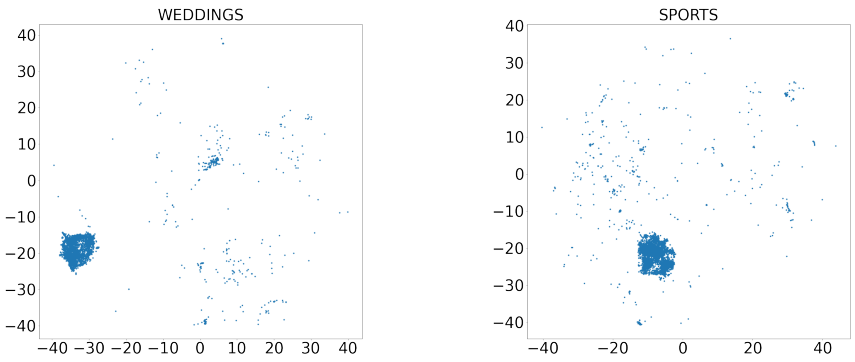
score increases significantly. Hence, we stop our training after 2 epochs to extract our embeddings for our t-SNE visualization.

Figure 6.3 already gives us some indication of overlaps between news categories i.e. "ENTERTAINMENT" and "COMEDY" overlap in a few places. However, as it is hard to visualize all overlaps in one single t-SNE plot, we compute individual t-SNE plots for all the news categories and compare them for inference. As we cannot show all individual plots here due to lack of space, we show it for one pair of similar and dissimilar plots.



(a) FOOD & DRINK (b) TASTE

Figure 6.4: FOOD & DRINK vs TASTE



(a) WEDDINGS (b) SPORTS

Figure 6.5: WEDDINGS vs SPORTS

We can see in Figure 6.4, the t-SNE representation of the news entries in the categories of "FOOD & DRINK" and "TASTE" show that there is high similarity in the data points whereas for two categories in Figure 6.5 like "WEDDINGS" and "SPORTS", indeed separate clusters are formed which do not overlap. Thus, t-SNE on our embeddings gives us a good indication of our problem of overlap and now we will proceed to develop a more robust algorithm which can show us class overlaps much better.

6.3 Algorithm detecting overlapping classes

We realised that for a robust mechanism to detect similarity between classes, we should decide on a statistical metric which can give us a measure of the similarity between distributions. In order to do this, we fit a probability distribution for each news category. We do this on the t-SNE vectors calculated on the output of our pre-classifier layer and fit a $2D$ histogram for each category. As we now have a probability distribution for each category, we compute the pairwise Jensen-Shannon divergence [22] between all the entries in news categories. Our overall algorithm is as follows :

Algorithm 1 Overlapping Class Detection Algorithm

1: Compute t-SNE on the output of the pre-classifier layer
2: **for** all categories of t-SNE vectors **do**
3: Fit a 2D histogram on the t-SNE vectors
4: Compute pairwise Jensen-Shannon (JS) divergence between the histograms
5: **end for**
6: Plot a confusion matrix for the JS divergence values
7: Fit a new histogram on the JS divergence values
8: From the JS divergence histogram, **select**
9: Threshold value for similarity to merge classes
10: From the JS divergence confusion matrix, **select**
11: Threshold value for similarity to remove classes
12: **return** possible classes for merging and removing

The measure of similarity we use for measuring similarity between our news categories is the Jensen-Shannon divergence. For two probability distributions P and Q, the Jensen-Shannon divergence is defined as :

$$JSD(P||Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q||M)$$

where

$$M = \frac{1}{2}(P + Q)$$

and $D(P||Q)$ is the Kullback–Leibler divergence given as

$$D(P||Q) = \sum_{x} P(x) \frac{P(x)}{Q(x)}$$

Here, the Jensen-Shannon divergence is similar to the Kullback–Leibler divergence for calculating the similarity between probability distributions, with the advantage of being symmetric and always returning a finite value. We choose this measure for these very reasons as it makes it easier to interpret for our use case.

Finally, one thing we ensure while fitting the $2D$ histograms is that we select the range of the distributions in such a way that it fits the histogram for every category of news, in the same $2D$ space. For this, we fix the range of the histograms in such a way that it is determined by the maximum and minimum values from the entire set of the combined t-SNE vectors.

Finally, after calculating our JS divergence matrix and histogram, we check for a *threshold value* of similarity, below which we decide with confidence that the categories can be merged into one category, as the news entries present in them are extremely similar. We also choose a threshold for removing categories which are extremely widespread i.e. their news entries are representative of many other specific classes. To do this, we select a *dropping threshold* and a *count of categories* which should be below this threshold, for a given category. We feed these values into our algorithm which finally returns the classes which can probably be removed from the dataset. We assert that our algorithm uses a statistical measurement as a basis to detect similarity and overlap between classes, however at this stage, it provides an indication than a definite proof and we leave it for future implementation to make this algorithm more robust.

Figure 6.6 shows the JS heatmap and figure 6.7 shows the histogram for JS divergence values for our algorithm. After analysing the values, we choose 0.69 as a threshold for merging classes and a 0.7 threshold with a count of at least 3 for removing a class. On feeding these thresholds for our dataset, we get the following output :

Possibly merge these classes :

BUSINESS – MONEY

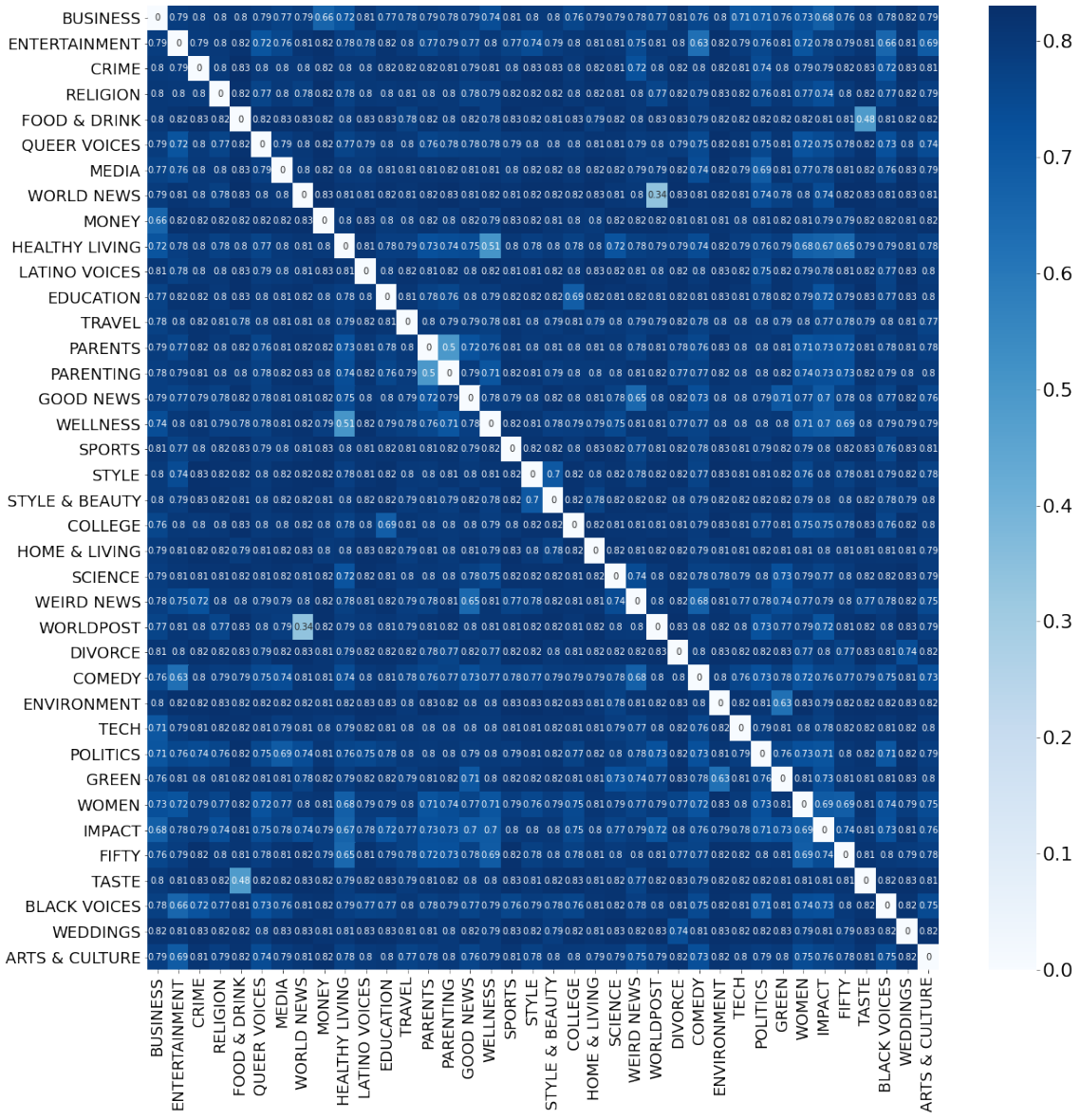


Figure 6.6: Pairwise JS divergence heatmap

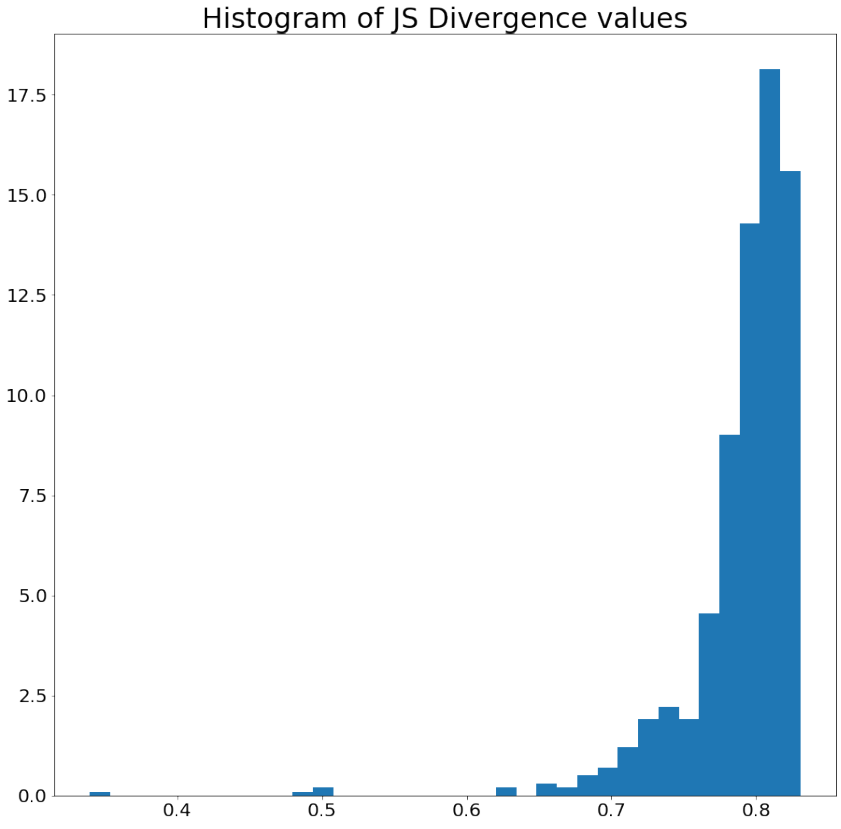


Figure 6.7: Histogram of JS divergence

ENTERTAINMENT – COMEDY
ENTERTAINMENT – BLACK VOICES
FOOD & DRINK – TASTE
WORLD NEWS – WORLDPOST
HEALTHY LIVING – WELLNESS
HEALTHY LIVING – WOMEN
HEALTHY LIVING – IMPACT
HEALTHY LIVING – FIFTY
PARENTS – PARENTING
GOOD NEWS – WEIRD NEWS
WEIRD NEWS – COMEDY

ENVIRONMENT – GREEN

Multiple overlap: Possibly drop these classes:

ENTERTAINMENT
HEALTHY LIVING
WELLNESS
WOMEN
IMPACT
FIFTY

We understand that setting the threshold for our algorithm is similar to hyper-parameter tuning and it cannot produce an output to merge or drop with 100% certainty. However, at this stage, it is a strong quantifiable justification of categories which can be merged or removed, which then needs to be fed into our model again, in order to determine the efficacy of the overall process.

6.4 Evaluation on an updated dataset

Based on the results of our class overlap detection algorithm, we decided to merge some of the categories, remove some categories and run our neural network model on the new dataset. When we merge categories in our dataset, we keep the final category label as the initial category which had the highest number of news entries before merging. In the new version of the dataset, we merge:

PARENTING + PARENTS $\rightarrow$ PARENTS
TASTE + FOOD & DRINK $\rightarrow$ FOOD & DRINK
HEALTHY LIVING + WELLNESS $\rightarrow$ WELLNESS
GREEN + ENVIRONMENT $\rightarrow$ GREEN
MONEY + BUSINESS $\rightarrow$ BUSINESS
COMEDY + ENTERTAINMENT $\rightarrow$ ENTERTAINMENT
COLLEGE + EDUCATION $\rightarrow$ EDUCATION
WORLDPOST + WORLD NEWS $\rightarrow$ WORLD NEWS

We remove the following classes as they were indicative to be too widespread in their content, and thus not being extremely representative. Hence, we remove :

FIFTY
IMPACT

After this process, we are left with 28 news categories as compared to the original 38 news categories. We run our model for 9 epochs again on this updated dataset, having reduced the problem of both class overlap and data imbalance between news categories. We calculate the same metrics as before on this new run, top prediction accuracy, top 3 predictions accuracy, mean F1 score and Mean Reciprocal Rank (MRR). In the training phase, we saw the epoch 6 had the best performance metrics, beyond that the model did not improve. We report those best values in our evaluation. Table 3 shows the summary of our results.

Table 3: Evaluating on the updated dataset

Performance Metric	**Value**
Accuracy (Top prediction)	73.60%
Accuracy (Top 3 predictions)	90.86%
Mean F1 Score	0.6590
MRR	0.8095

We see that we have been able to obtain a significant improvement in our model's performance, after we updated the dataset's news categories based on the output of our algorithm. We see that the top prediction accuracy increases to 73.60% from the original base value of 65.67% We also calculate the top three prediction accuracy as our problem is multi-class with similar classes, the top three accuracy increases to 90.86%. We see that the model is able to learn the top class with greater accuracy compared to relying on predicting the top 3 classes. We attribute this to proper reduction and merging of classes based on our algorithm. We also see a significant improvement in our mean F1 score, which increases to 0.6590, which means our model confuses less between the classes now. Finally an improvement in the MRR indicates that the top prediction is much higher than it was in the original model. The extent of improvement of the model here will depend on an extent to the degree of reduction we choose to allow in the number of dataset classes, which depends on the user.

Chapter 7

Conclusion

In our project, we attempt to build a neural network model which can classify a complex news dataset. In order to do this, we experiment with both an RNN based classification model and a transformer based classification model, after which we end up choosing the transformer based BERT model based on it's architectural advantages and ease of use. After our preliminary run of the model on our dataset, we realise a fundamental problem in our dataset. We have several news categories, which overlap in context. Hence, news entries from these categories are often confused with news entries from a similar category, as our model learns contextual embeddings for the news entries, which are very similar across these categories. Along with this, we have a huge imbalance in the number of news entries in every category. We propose a new algorithm to learn similarity between news categories, based on t-SNE dimensionality reduction of vectors and Jensen-Shannon divergences calculated on them, which in turn give us an indication of categories which should be merged due to similarity and categories which should be removed due to lack of specificity. Finally, we test our model on an updated dataset, which we reduced based on our algorithm, and show it is possible to achieve good performance improvement with our overall model.

Bibliography

[1] A. Rajaraman and J. D. Ullman, "Data mining," *Mining of Massive Datasets*, pp. 1–17, December 2011. [Online]. Available: https://doi.org/10.1017/CBO9781139058452.002

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," January 2013. [Online]. Available: https://arxiv.org/pdf/1301.3781.pdf

[3] J. Pennington, R. Socher, and C. Manning, "Data mining," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1–17, October 2014. [Online]. Available: https://www.aclweb.org/anthology/D14-1162

[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," june 2017. [Online]. Available: https://arxiv.org/pdf/1607.04606.pdf

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," October 2018. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf

[6] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," August 2019. [Online]. Available: https://arxiv.org/pdf/1908.10084.pdf

[7] D. Cer, Y. Yang, S. yi Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Universal sentence encoder," March 2018. [Online]. Available: https://arxiv.org/pdf/1803.11175.pdf

[8] O. Fuks., "Classification of news dataset," 2018. [Online]. Available: http://cs229.stanford.edu/proj2018/report/183.pdf

[9] D. Cameron-Steink., "Short-text classification using unsupervised keyword expansion," 2019. [Online]. Available: https://arxiv.org/pdf/1909.07512.pdf

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, August 2009. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5206848

[11] R. Misra., "News category dataset," 2019. [Online]. Available: https://www.kaggle.com/rmisra/news-category-dataset

[12] "Huffpost." [Online]. Available: https://www.huffpost.com/

[13] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," july 2018. [Online]. Available: https://arxiv.org/pdf/1705.02364.pdf

[14] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, p. 972–981, December 2017. [Online]. Available: https://dl.acm.org/doi/pdf/10.5555/3294771.3294864

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," December 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf

[16] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," December 2019. [Online]. Available: https://arxiv.org/pdf/1910.01108.pdf

[17] "Transformers." [Online]. Available: https://huggingface.co/transformers/index.html

[18] "Pytorch." [Online]. Available: https://pytorch.org/

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," December 2014. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

[20] "Scikit-learn." [Online]. Available: https://scikit-learn.org/stable/

[21] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 27, pp. 2579–2605, November 2008.

[22] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 27, p. 145–151, November 1991.