Advanced Natural Language Engineering Coursework (G5114)

Candidate: 246504

1 Introduction

In the field of Natural Language Processing (NLP), there are a plethora of applications in which it can be applied in order to gain better insights/achieve better things from language. In this report, 2 approaches were experimented on 2 separate tasks involving propaganda.

1.1 Propaganda

Propaganda is a technique that strives to influence the minds of unsuspecting people into following a certain agenda (Martino et al., 2020). Therefore, it is most successful when it remains undetected by the reader. For this reason, the creation of algorithms/models that can detect when propaganda is being used, as well as discern the specific type of propaganda at play, have paramount importance in enabling the public to fight back against mental manipulation.

1.2 Tasks

The objective of the first task was to create a model that can accurately predict whether a sentence contains propaganda, i.e. a **binary classification task**. Task 2 was to define a model which, when given a snippet of propaganda, can classify specifically what form of propaganda is being used, i.e. **multi-class**.

1.3 Dataset

The dataset in use is made up of sentences, all of which are paired with a label indicating the propaganda content. The 9 labels are:

1. appeal_to_fear_prejudice
2. casual_oversimplification
3. doubt
4. exaggeration,minimisation
5. loaded_language
6. flag_waving
7. name_calling,labeling
8. repetition
9. not_propaganda

The first of the 8 labels are a subset of the propaganda techniques identified in the Propaganda techniques Corpus (Martino et al., 2020), and the last is a simple marker to indicate no propaganda. Within these sentences are snippets where the potential propaganda lies, which are marked with a '<BOS>' and '<EOS>' token. The dataset is split into training and validation data, which contain 2560 and 640 samples respectively.

1.4 Models

There were 2 models that were applied to each task.

Naïve Bayes: This is a probabilistic model based on Bayes' theorem. It is relatively simple compared to other classification methods, and its results usually reflect as such (Zhang and Oles, 2001).

Fine-tuned BERT: In terms of complexity, this method is the other end of the spectrum compared to Naïve Bayes. BERT is a model pre-trained on masked language modelling and next sentence prediction tasks in order to learn contextualised bi-directional representations for words and sentences. From there, it can be fine-tuned to a variety of tasks including sequence classification (Devlin et al., 2018).

2 Methodology

This section will cover the theory used for evaluating the proposed solutions to the two tasks, as well as the theory for the solutions themselves. Furthermore, it will specifically cover how these approaches were applied to the two tasks.

2.1 Performance Metrics (Grandini et al., 2020)

This section will be a review of the relevant elements from a review conducted by Grandini et al. (2020), in which they comprehensively cover the various performance measuring techniques for classification. There are several scores that can be calculated to represent different aspects of classification when evaluating performance. The 4 metrics that will be used to evaluate the models will be: Accuracy, F1-Score, Precision, Recall.

2.1.1 Confusion Matrix

Calculating these metrics is made much simpler when the predictions are visualised using a **confusion matrix**.

		Predicted Class	
		P	N
True class	P	16	5
	N	10	9

Figure 1: Dummy confusion matrix of binary classification problem

Figure 1 displays an example of a confusion matrix. 'P' denotes 'positive', and 'N' denotes 'negative'. In this case, these would equate to 'propaganda' and 'not propaganda'. Before calculating, the scores listed previously, 4 quantities must be counted:

- True Positives (TP): Positive samples that are predicted as such, intersection of row and column P
- False Positives (FP): Negative samples that are predicted as positive, intersection of column P and row N
- True Negatives (TN): Negative samples that are predicted as such, intersection of column and row N
- False Negatives (FN): Positive samples that are predicted as negative, intersection of column N and row P

2.1.2 Accuracy

The most straightforward performance metric used in this report was **accuracy**. It's calculated simply as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 2: Accuracy Formula

Accuracy score represents the proportion of the model's predictions that are correct.

2.1.3 Precision & Recall

These 2 scores are more specific insights into how the model is predicting.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

Figure 3: Precision & Recall Formulae

Precision is a proportional measure of how many positive (i.e. propaganda) predictions the model gets correct, i.e. how precise it is being. On the other hand, recall is the proportion of positive predictions out of the total positives in the dataset, i.e. how thoroughly it is predicting.

2.1.4 F1-Score

F1-Score acts as a valuable insight when evaluating the overall performance of a model as well as accuracy, and is calculated as a weighted average of precision and recall, using the concept of harmonic mean.

$$F1 - Score = \frac{2}{Precision^{-1} * Recall^{-1}} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 4: F1-Score Formula

2.1.5 Multi-class scoring (Task 2)

Up to now, the descriptions of the previous scores have been from the perspective of binary classification. When performing multi-class classification, the scores can be calculated either across the whole model, or within each class.

		Predicted			
		A	B	C	D
True	A	5	2	3	2
	B	3	9	2	1
	C	1	2	7	1
	D	1	1	2	8

Figure 5: Mutli-class confusion matrix

To calculate the scores for each class, TP, FP, TN, and FN scores must be quantified for them individually. Using class A as an example:

- TP: Intersection of column and row A (5)
- FP: Total of column A - TP (10 - 5 = 5)
- TN: Total of all cells - TP - FP - FN (50 - 5 - 5 - 7 = 33)
- FN: Total of row A - TP (12 - 5 = 7)

Once these quantities are counted for all of the classes, Precision, recall and F1-Score can be calculated for each class, and the entire model. If using **micro averaging**, the TP, FP, TN, and FN scores of all the classes are summed in order to calculate the model's precision, recall and F1. However, if using **macro averaging**, the models precision, recall and F1 scores are calculated using the mean of each score across the classes respectively. As an example for both methods, here are the equations for precision using micro and macro averaging.

$$N = Number\,of\,classes \qquad TP_i = True\,positives\,of\,class\,i$$

$$Micro\,averaged\,precision = \frac{TP_1 + TP_2 ... + TP_N}{TP_1 + TP_2 ... + TP_N + FP_1 + FP_2 ... + FP_N}$$

$$Macro\,averaged\,precision = \frac{Precision_1 + Precision_2 ... + Precision_N}{N}$$

Figure 6: Micro vs Macro averaged precision

When using micro averaging, accuracy, precision, recall and F1-score are all equal. Therefore, to provide a more complete insight into performance, macro averaged was used in this investigation.

2.2 Naïve Bayes (Raschka, 2014)

The first of the two models used in this report was Naïve Bayes (NB). It is a probabilistic model based on Bayes' Theorem. Raschka (2014) presented the theory behind a NB model as well as its application for text classification, and this section will review the contents of their report. It performs classification by assigning a given sample the class which returns the maximum **posterior probability**. The *Naïve* adjective comes from its assumption that all the features are *independently and identically distributed (i.i.d)*. This is seldom the case in reality, however the model can still achieve respectable results.

2.2.1 Bayes' Theorem

Bayes' theorem is the theory of posterior probability. This can be understood as the probability

of observing a class given a data sample.

Let:

- **Evidence**: x_i be the feature vector of sample i, $\quad i \in \{1, 2, 3...n\}$
- **Prior probability**: w_j be class $j \quad j, \in \{1, 2, 3...m\}$
- **Conditional probability**: $P(x_i|w_j)$ is the probability of observing feature vector x_i given class w_j

$$P(w_j|x_i) = \frac{P(x_i|w_j) * P(w_j)}{P(x_i)}$$

The formula above is what a NB model uses to classify data. The prior probabilities for each class, as well as the evidence, are learnt during training on train data. Then, when classifying unseen data, the class that maximises a sample's posterior probability is assigned to each sample.

2.2.2 Bag-of-words

When representing a sample of text for a NB classifier, a bag-of-words model based on the training corpus must be constructed. This involves defining a **vocabulary**: A collection of all the different words that occur in the corpus. Using this, feature vectors can be constructed to represent each sample, which are usually made up of frequency values representing the occurrences of each word in the sample.

Consider this sentence:

I drive cars, I don't ride bikes.

A vocabulary defined from this sentence, and the sentences feature vector, would look like this:

{I:0, drive:1, cars:2, don't:3, ride:4, bikes:5}
[2, 1, 1, 1, 1, 1]

As there are 2 occurrences of the word 'I', its corresponding frequency value in the vector is 2, and 1 for every other word. Now consider a new sentence and its feature vector:

I drive cars all day.
[1, 1, 1, 0, 0, 0]

As the vocabulary does not contain the words 'all' and 'day', the feature vector for the sentence above cannot represent them.

2.2.3 Preprocessing

In building a NB model for both tasks, different preprocessing methods were implemented and compared against each other. Preprocessing prefers to operations performed on the text prior to the model learning process, and includes the bag-of-words generation.

Tokenisation: A mandatory first step is tokenisation. This refers to taking a sample of text, and converting it into a list of tokens that usually represent individual words.

I like to drive.
['I', 'like', 'to', 'drive.']

Notice that the punctuation and capitalisation remains unchanged in the tokens.

Punctuation removal & capitalisation normalisation: This is the first technique that will be experimented with in both tasks. In order to normalise the text so that a vocabulary can be generated, there is the option to remove punctuation from the tokens, as well as convert all the letters to lower/upper case.

['i', 'like', 'to', 'drive']

Doing this ensures that when the vocabulary is generated, matching words that have different capitalisation, or surrounding punctuation, don't get marked as different. The models were evaluated with and without this technique.

Stop-word removal: When creating a vocabulary from a large corpus, the resulting feature vectors can be very large and contain a lot of zeroes representing the words absent in the sample. One way to combat this is to remove the uninformative words (stop-words) from the samples (Silva and Ribeiro, 2003). These include: 'and', 'to', 'it' etc. This can also help the model to pay more attention to the more informative words.

TF-IDF vectors: Rather than using simple frequency values in the feature vectors, TF-IDF (Aizawa, 2003) values can be used to represent the word content within the samples. The formula is as follows:

- Term Frequency $(TF) = \dfrac{Number\ of\ occurrences\ of\ current\ word\ in\ given\ sample}{Total\ number\ of\ words\ in\ document}$
- Inverse Document Frequency $(IDF) = \log(\dfrac{Number\ of\ documents}{Number\ of\ documents\ containing\ current\ word})$

$$\text{TF-IDF} = TF * IDF$$

During vocabulary construction, the IDF values for each word are calculated. Then, when the feature vectors are generated, TF-IDF values are used in place simple frequency values to represent each word in the vector.

2.3 Fine-tuned BERT, (Devlin et al., 2018)

The second approach used for each task was a fine-tuned Bidirectional Encoder Representations from Transformers (BERT) model. Devlin et al. (2018) introduced this language representation model and covered both its pre-training process as well as its possible applications. BERT is a large language model (LLM) that has been pre-trained to perform two main tasks on large unlabelled corpora: BooksCorpus(800M words), Wikipedia(2,500M words). First, it was trained on a Masked Language Modelling (MLM) task, in which it learns bidirectionally contextualised word embeddings to represent words. Then, it is trained to complete a Next Sentence Prediction (NSP) task. From here it can be fine-tuned for various tasks i.e. transfer learning (Torrey and Shavlik, 2010), including **sequence classification**. This approach holds two main advantages over a NB approach: It can represent semantic similarity between different words, and represent and process sequential information i.e. word order.

2.3.1 MLM pre-training

Masked language modelling, also know as the *Cloze* task (Taylor, 1953), involves replacing the target word with a mask token. In this case, BERT then uses the words (context) to the left and right of the masked word to predict what the masked word is. The pre-training process therefore involves optimising these representations so that the word is correctly predicted as often as possible. As words are not masked during fine-tuning, certain steps were taking to prevent overfitting to this task. Target words are only replaced with a mask-token 80% of the time. 10% of the time, it is replaced with a random token, and the final 10% the word remains unchanged.

Contextualised Word Embeddings: This task allows for the model to create contextualised word embeddings. This refers to a vector representation of words that refer to its position in n dimensional space.

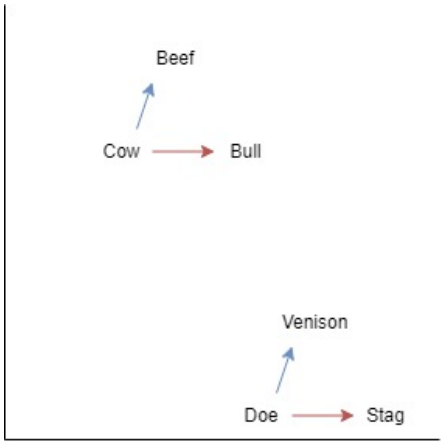

Figure 7: Example Word embeddings

Figure 7 displays a some words embedded in 2-dimensional space. The main characteristic of these embeddings is that words with similar meaning exist nearby in this space. Furthermore, pairs of words that share similar relationships have similar distances between the words.

2.3.2 NSP pre-training

This task is a binary task, in which the model is given two sentences: A and B. 50% of the time, B is actually the next sentence after A, labelled as *IsNext*. The rest of the time, its a random sentence from the rest of the corpus labelled as *NotNext*. The objective of this task is to predict the label of B, teaching the model to understand the **relationship between sentences**.

2.3.3 Tokenisation and Encoding

Based on the learned word and sentential embeddings, BERT provides its own tokeniser. It works by converting known words into IDs which correspond to a token, segment, and positional embedding which combine to convey the semantic meaning within the sentences and samples. For words that are more uncommon, it splits the word into a token of its morphological root, and a second subword token which will be converted to a character based representation. For example, take the word 'igneous'. BERT tokeniser would split it into the token 'ign' and '#neous', with the hash symbol indicating a subword token.

Tokeniser hyper-parameters: In order to ensure uniformity in the data samples, both *padding* and *truncation* were set to 'True' to ensure the all encodings were the same length. The max_length value (to which samples would be padded/truncated) was set to 220. This value was decided based on a combination of the longest sample (143 words), and Zipf's law which dictates that approximately half the words in a corpus will only occur once (Corral et al., 2015). As the tokeniser will split uncommon words into 2 sub-tokens, it was estimated that the longest sample would have roughly 214 tokens. Hence the choice of 220 for the max length.

2.3.4 Fine-tuning

The BERT model can be fine-tuned to perform a large selection of NLP tasks. To perform sequence classification, it involves adding densely connected layers (Dongare et al., 2012) on top of the final pooling (Gholamalinezhad and Khosravi, 2020) layer of the pre-trained BERT. Then, the model learns to classify through iteratively making predictions, computing the **loss** between those predictions and the correct answers, and altering the parameters in the model based on their gradient with respect to the loss (Werbos, 1990).

Loss: There are various loss functions commonly used in machine learning (Wang et al., 2020). For classification, a popular choice is **cross-entropy loss** (Zhang and Sabuncu, 2018).

- M = Number of classes
- c = Current class
- o = Current observation (sample)
- y = Binary indicator that c is the correct class for observation o
- p = Predicted probability that o is in class c

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Backpropagation & gradient descent: Using the calculated loss, backpropagation (Werbos, 1990) is performed on the model in order to get the gradient of each parameter with respect to the loss. Then, the value of these parameters are changed in the opposite direct of their respective gradients by a particular value (specified by learning rate), in a process called **stochastic gradient descent** (Amari, 1993), with the objective of improving the predictions.

2.3.5 Hyper-parameters

When fine-tuning BERT, various hyper-parameters are defined which dictate certain properties about the training process. In task 1, the effect of these hyper-parameters were investigated:

- **Batch size**: Number of samples used during each step of training
- **Epochs**: Number of full passes the training dataset makes through the network during training

Below are the notable hyper-parameters that remained fixed during training:

- **Weight Decay = 0.01**: A form of regularisation in which large parameters are reduced to discourage overfitting (Bos and Chug, 1996)
- **Learning Rate = 5e-5**: The value that parameters are adjusted by during stochastic gradient descent.

Regarding task 2, there was a clear issue of overfitting present for each iteration. Therefore, these regularisation techniques were explored:

- **Batch size**
- **Weight Decay**
- **Dropout:** A probability for neurons within given layers to be turned off at random during a forward pass. Helps to prevent dependencies, therefore reduce overfitting

2.4 Task 1: Propaganda detection

The objective of this task is for a model to be able to predict whether a given sentence contains propaganda or not. It is therefore a binary classification task. Prior to the model specific preprocessing techniques that were evaluated for this task, the sentences had their '<BOS>' and '<EOS>' tags removed, so that they reflected the composition of a standard sentence. Regarding the labels, a list of binary indicators was generated. The label 'not_propaganda' was associated with a 0, and the rest of the labels were associated with a 1.

2.5 Task 2: Propaganda Classification

The objective for the second task was for a model, given a snippet of text known to contain propaganda, to be able to predict what kind of propaganda it contained. To extract these snippets from the original dataset, the sentences know to contain propaganda were separated using the list of binary indicators defined for the previous task. Then the text around the snippet was removed leaving just the snippet of propaganda. Finally, each snippet was associated with a number from 0-7, indicating the type of propaganda contained:

1. appeal_to_fear_prejudice
2. causal_oversimplification
3. doubt
4. exaggeration,minimisation
5. loaded_language
6. flag_waving
7. name_calling,labeling
8. repetition

3 Results

This section will cover the results of the different experiments conducted with each model, as well as highlighting the best version of each.

3.1 Task 1

3.1.1 Naïve Bayes

For all iterations of Naïve Bayes which were experimented with, the data went under these processes, in addition to that which was being tested:

1. Tokenisation
2. Case Normalisation
3. Vocabulary construction
4. Feature vector generation

As a baseline, a model with no additional techniques was tested. The scores were as follows:

- Accuracy = 67.813%
- F1-Score = 72.386%
- Precision = 61.785%
- Recall = 87.379%

Judging by the high recall and the relatively low precision, the model seemed to be over-classifying the sentences as propaganda. This could be due to the fact that with the training dataset, 50.43% of the samples contain propaganda. Whereas, in the validation set, only 48.28% of the sentences contain propaganda. Therefore, as NB models rely on the observed probability of each class occurring, it is slightly over-classifying the validation set.

3.1.1.1 Removing punctuation and stop-words: The next experimentation involved removing punctuation and stop-words. This was done in the hopes that removing arguable redundant information would improve the models performance.

- Accuracy = 68.75%
- F1-Score = 71.347%
- Precision = 64.01%
- Recall = 80.583%

While accuracy has increased by approximately 1 percentage point, F1-Score has in turn decreased by 1. Precision and recall changed slightly more significantly, precision increased by 2.5 percentage points, and recall decreased by nearly 7. As accuracy increased using this method, it was used in conjunction with the next two methods.

3.1.1.2 Using snippets for training: While the task was to classify entire sentences, the parts of the sentences that contained propaganda were still marked. Due to this, an experiment was conducted in which the vocabulary was generated using only the contents of the snippets within the training set. The feature vectors were still generated from the full sentences, but naturally the words that didn't occur in the snippets were not represented.

- Accuracy = 68.28%
- F1-Score = 70.707%
- Precision = 63.802%
- Recall = 79.288%

This technique resulted in decreases in performance across the board. Its possible then that while the snippets are where the propaganda lies, the words around the snippets contained information that was helpful in discerning if propaganda would be or had been used in the sentence.

3.1.1.3 TF-IDF vectors: The final NB iteration used for this task involved TF-IDF vectors. For this, vocabulary generation was reverted back to using the entirety of each sentence, rather than just the snippet.

- Accuracy = 69.375%
- F1-Score = 71.345%
- Precision = 65.067%
- Recall = 78.964%

Out of all of 4 iterations, the accuracy is the highest here. F1-Score is still lower than the first iteration by 1 percentage point, however this model still performs better overall.

3.1.2 BERT

For the first iteration, a batch size of 64 was used for 3 epochs.

- Accuracy = 85%
- F1-Score = 83.505%
- Precision = 89.011%
- Recall = 78.641%

Already, this model is performing much better than any of the NB iterations. The next iterations were run with altered batch sizes and for different numbers of epochs.

3.1.2.1 32 Batch size, 3 Epochs: This iteration was run a batch size of 32 rather than 64. While this will cause longer training times per epoch, reducing batch size can help models to generalise to unseen data (Masters and Luschi, 2018).

- Accuracy = 94.844%
- F1-Score = 94.686%
- Precision = 94.231%
- Recall = 95.146%

This did result in a much better performance on validation data, with all scores being above 94%.

Step	Training Loss	Validation Loss	Accuracy
60	0.277300	0.275316	0.878125
120	0.195000	0.192973	0.940625
180	0.071700	0.226782	0.935937
240	0.048500	0.189649	0.948438

Figure 8: Logs for 32 Batch size, 3 Epochs

Figure 8 indicates that the loss may not have reached a minimum yet. Therefore, the model may benefit from more training epochs.

3.1.2.2 32 Batch Size, 10 Epochs: This model had an equal batch size to before, but was trained for over triple the epochs.

- Accuracy = 93.594%
- F1-Score = 93.376%
- Precision = 93.226%
- Recall = 93.528%

The models performance has actually worsened slightly, which is unexpected considering the model is set to load its best version after training. The reason why is indicated in the logs.

Step	Training Loss	Validation Loss	Accuracy
60	0.157400	0.338036	0.920312
120	0.126900	0.263896	0.926562
180	0.086100	0.330745	0.937500
240	0.073400	0.338620	0.939063
300	0.122400	0.244833	0.942187
360	0.039200	0.351610	0.934375
420	0.037500	0.255032	0.956250
480	0.015300	0.241805	0.957812
540	0.002100	0.274536	0.946875
600	0.014100	0.286277	0.950000
660	0.000300	0.314957	0.948438
720	0.010200	0.309125	0.951562
780	0.000300	0.308885	0.950000

Figure 9: Logs for 32 Batch size, 10 Epochs

Figure 9 shows that there are several iterations that achieved a higher accuracy than the model which was returned after training. However, that with the lowest validation loss was that which the model considered the best, so it was returned even though it had a worse accuracy. Despite this, the models with the best accuracy and validation loss were still reached before the end of the fifth epoch, therefore the next iteration experimented with 5 epochs.

3.1.2.3 16 Batch Size, 5 Epochs: As well as reducing the number of epochs, this iteration saw a batch size reduced by 50% yet again. This was to see if the previous success found in this technique could be furthered.

- Accuracy = 95.781%
- F1-Score = 95.652%
- Precision = 95.192%
- Recall = 96.117%

This technique has again resulted in improvement across the board, and the best solution for this task. Furthermore, the logs show that this version of the model was reached relatively early.

Step	Training Loss	Validation Loss	Accuracy
60	0.157400	0.338036	0.920312
120	0.126900	0.263896	0.926562
180	0.086100	0.330745	0.937500
240	0.073400	0.338620	0.939063
300	0.122400	0.244833	0.942187
360	0.039200	0.351610	0.934375
420	0.037500	0.255032	0.956250
480	0.015300	0.241805	0.957812
540	0.002100	0.274536	0.946875
600	0.014100	0.286277	0.950000
660	0.000300	0.314957	0.948438
720	0.010200	0.309125	0.951562
780	0.000300	0.308885	0.950000

Figure 10: Logs for 16 Batch size, 5 Epochs

The logs displayed in figure 10 show that the best model was found at step 480, which is at the beginning of the fourth epoch.

3.2 Task 2

3.2.1 Naïve Bayes

Like before, the first iteration that was tested for this task only used the 4 standard preprocessing techniques, preceded by the snippet extraction process. The results of this iteration were as follows:

- Accuracy = 39.482%
- F1-Score = 37.804%
- Precision = 52.240%
- Recall = 39.234%

The results of this model are much worse than on the previous task, with the only score reaching above 50% being precision. Clearly, with the introduction of more classes, NB seems to be struggling to make predictions with its probabilistic nature.

3.2.1.1 Punctuation and Stop-word removal: Given the insignificant nature of this technique, the change in results was not expected to be large.

- Accuracy = 39.482%
- F1-Score = 38.266%

- Precision = 46.220%
- Recall = 39.116%

This iteration has identical accuracy to the previous. However, F1-Score has increased by a small amount, indicating a slightly better average performance across the individual classes. Overall the classifier is still performing poorly.

3.2.1.2 TF-IDF Vectors: The final iteration of NB that was tested was with TF-IDF vectors. Given the performance up to this point on this task, expectations were not high regarding the results this would produce.

- Accuracy = 43.042%
- F1-Score = 42.606%
- Precision = 47.677%
- Recall = 43.065%

While this performance is nothing special overall, it is still a significant improvement over the previous 2 iterations. Accuracy, f1-score, and recall have all increased by roughly 4 percentage points.

3.2.2 BERT

For the first iteration, Batch Size was reverted to 32 again, and training took place for 5 Epochs.

- Accuracy = 62.140%
- F1-Score = 61.668%
- Precision = 62.455%
- Recall = 62.012%

These scores are much better than those from any of the NB iterations. However, it does still pale in comparison to BERT's performance on task 1.

Step	Training Loss	Validation Loss	Accuracy
60	1.682100	1.396373	0.488673
120	0.982900	1.115628	0.618123
180	0.524800	1.084315	0.621359

Figure 11: Logs for 32 Batch size, 5 Epochs

According to the figure 11, there was a large discrepancy between the training and validation loss. Therefore, some regularisation could've been beneficial.

3.2.2.1 16 Batch Size, 5 Epochs: This iteration was trained with a batch size of 16, like in task 1. Furthermore, the number of steps between validation was decreased in order to show more frequent insights into the training process.

- Accuracy = 61.165%
- F1-Score = 61.143%
- Precision = 64.696%
- Recall = 60.878%

This seemed to slightly decrease performance, and still the issue of overfitting was present.

Step	Training Loss	Validation Loss	Accuracy
80	0.116600	1.734236	0.611650
160	0.123400	1.966767	0.627832
240	0.117000	2.008120	0.608414
320	0.058800	1.947041	0.637540
400	0.048100	1.994384	0.627832

Figure 12: Logs for 16 Batch size, 5 Epochs

Judging by the disparity between training and validation data in figure 12, overfitting was still a glaring issue. Therefore, another form of regularisation was required.

3.2.2.2 32 Batch Size, Weight Decay 0.015, 10 Epochs: This iteration saw an increase in weight decay in an attempt to reduce the overfitting present in previous iterations.

- Accuracy = 61.149%
- F1-Score = 60.917%
- Precision = 61.767%
- Recall = 61.783%

The results here are mostly the same if not worse than before. Overfitting still seemed to be the limiting factor in any progress that BERT can make on this task.

3.2.2.3 Dropout introduction: In this final iteration, the hyper-parameters were the same as the previous, but this time the model was configured with new dropout values:

- Hidden dropout = 0.25
- Attention Probabilities dropout = 0.25
- Classifier Dropout = 0.5

The resulting scores were as follows:

- Accuracy = 61.812%
- F1-Score = 61.495%
- Precision = 62.027%
- Recall = 61.481%

While this model did perform better than the iteration with weight decay, it still did not perform as well as the initial version with no altered forms of regularisation.

Step	Training Loss	Validation Loss	Accuracy
80	1.900900	1.676046	0.330097
160	1.497100	1.309475	0.508091
240	1.041000	1.167139	0.566343
320	0.757600	1.145074	0.618123
400	0.585800	1.158513	0.618123

Figure 13: Logs for Dropout introduction

While the onset of overfitting in figure 13 is delayed compared to figure 11, the model still does not perform as well overall.

4 Conclusion

4.1 Task 1

Regarding the first task, both models proved to be somewhat viable solutions based on their performance. BERT performed much better across the board, but this was to be expected considering its extensive pre-training (Devlin et al., 2018), as well as its ability to represent sequential and semantic information. However, this is not to say that Naïve Bayes performed particularly badly.

4.2 Task 2

For the following task, neither model performed quite as well as on task 1. Naïve Bayes struggled to break 40% scores until TF-IDF vectors were introduced. BERT on the other hand did manage to get a respectable best performance of 62% accuracy, however this is still far less than how it performed on the binary task.

4.3 Further Work

Considering the high performance of BERT on task 1, it would be hard to make significant improvements on it with future optimisations. However, the results of task 2 showed that a lack of generalisability was severely limiting performance. Therefore, one avenue of further investigation could be to delve deeper into different types of regularisation (Kukačka et al., 2017). While BERT is considered a significant advancement in NLP (Torfi et al., 2020), there are a plethora of alternative methods for sequence classification, and another direction for further work could be to apply these methods to these tasks and weigh up how they compare.

References

Aizawa, A. (2003). An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65.

Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196.

Bos, S. and Chug, E. (1996). Using weight decay to optimize the generalization ability of a perceptron. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 241–246. IEEE.

Corral, Á., Boleda, G., and Ferrer-i Cancho, R. (2015). Zipf's law for word frequencies: Word forms versus lemmas in long texts. *PloS one*, 10(7):e0129031.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dongare, A., Kharde, R., Kachare, A. D., et al. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194.

Gholamalinezhad, H. and Khosravi, H. (2020). Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*.

Grandini, M., Bagli, E., and Visani, G. (2020). Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*.

Kukačka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*.

Martino, G., Barrón-Cedeno, A., Wachsmuth, H., Petrov, R., and Nakov, P. (2020). Semeval-2020 task 11: Detection of propaganda techniques in news articles. *arXiv preprint arXiv:2009.02696*.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

Raschka, S. (2014). Naive bayes and text classification i-introduction and theory. *arXiv preprint arXiv:1410.5329*.

Silva, C. and Ribeiro, B. (2003). The importance of stop word removal on recall values in text categorization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 1661–1666. IEEE.

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433.

Torfi, A., Shirvani, R. A., Keneshloo, Y., Tavaf, N., and Fox, E. A. (2020). Natural language processing advancements by deep learning: A survey. *arXiv preprint arXiv:2003.01200*.

Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global.

Wang, Q., Ma, Y., Zhao, K., and Tian, Y. (2020). A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Zhang, T. and Oles, F. J. (2001). Text categorization based on regularized linear classification methods. *Information retrieval*, 4:5–31.

Zhang, Z. and Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31.