Neural Networks Project

Fake News Classification

Adriana Haralambieva
Isabelle Kampono
Jonas Scholz
Maria Stateva

Neural Networks
Artificial Intelligence
Rijksuniversiteit Groningen
July 1st 2022

Contents

Abstract

Fake news has increasingly become a more concerning problem in the past few years with the influence of social media and the increase of information resources. However, classifying and verifying all that information manually can be difficult and time-consuming, or even inaccurate. This is a very important problem as it can easily manipulate people into a specific opinion or belief, without them even realising. That is why the purpose of this project is to model a fake news classifier. We made a text classifier using a feedforward neural network in order to perform this classification and it was able to classify the news titles with 91% accuracy.

1 Introduction

Fake news is a very prevalent and important issue in our society today. Considering how much media consumption there is and how much news affects our lives, this is a significant issue that everyone should take into consideration. The problem is that it is very hard for us as humans to accurately distinguish fake news from real news. However, machine learning and text classification can help us sort out the news we consume more accurately and efficiently, which is what we will try to do with this project.

Classification with a neural network is the process of analysing data in a way that predicts the class of all the output data points. In our project, we use the inputs of the article titles for our dataset and we try to sort them into two classes, fake or not fake. For the process of classification to take place we are trying to construct a function that would output the class of the testing inputs.

The data set that we are using was taken from Kaggle (Shahane, 2021). The dataset itself includes three columns one that has the title of all the articles one that contains the text of the articles themselves and the last one indicates the class of the article, either fake or not. Specifically, that is indicated with a 0 or 1, where 0 indicates that the article is fake and 1 indicates that the article is real. The whole dataset consists of 72,134 news articles with 35,028 real and 37,106 fake news ones. The dataset we used was actullly made from four different datasets combined to provide enough data for neural network training and minimize over-fitting.

Our main goal with this project is to get as close as possible to creating a model that would work and give somewhat of an accurate prediction. The main intention is to learn and compare different strategies and functions to the model and experiment with what works best with the goal in mind. We do not expect to construct a revolutionary or perfectly efficient model, we are more concentrated on the process of reaching a satisfying output rather than implementing the best model possible.

Because of our ambitious goal of creating this classifier we decided to base it on something we are familiar and comfortable with. For this reason, we decided to use a feedforward network for our model. We chose this network because we considered it the easiest to create and the network we knew the most about and we were most confident to work on. We also considered trying to create a second recursive neural network, however, the workload from the first model didn't give us the opportunity or time to create a second more complex model.

To test the success and performance of our we would mainly rely on the accuracy percentage for both our testing and validation data. To get the percentage of accuracy after every epoch we use both a cross-validation method as well as the k-folds cross-validation method. Another measure of success we implemented was to test a specific title or sentence which we thought or knew was fake or not and use it as an input for our model. This was a more subjective way of measuring our success with the project, but the results were sufficient enough and satisfying.

2 Data

We were able to find a dataset from Kaggle that has news article titles and texts, as well as a label that signifies whether that article is a fake news article or a real one. The dataset has 72134 entries, with 35028 fake news entries and 37106 real news entries. For this project, we are going to focus on using the titles of the articles to predict

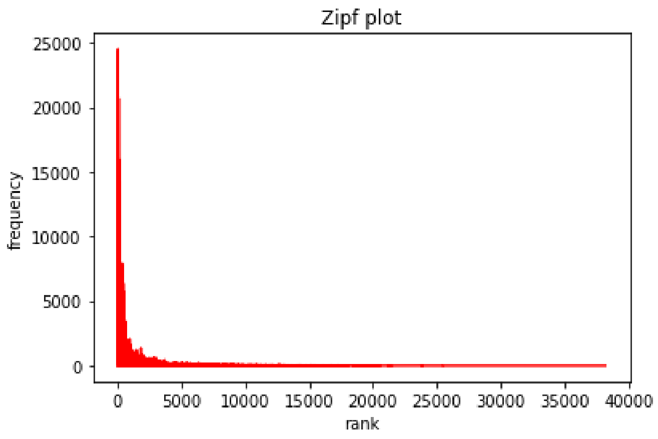

Figure 1: Frequency distribution of title words following Zipf's law (Lestrade, 2017)

whether it is a fake/real news title. The titles of the articles have a large range of words, ranging from 1-72 words in each title, some of which have out-of-vocabulary ('OOV') phrases. Most of the titles of the news articles are not standardized, meaning that they contain capitalized letters, punctuation, as well as numbers. Each news title has its own respective label, where 0 indicates a fake news title and 1 indicates a real news title.

Considering that we are working on a Natural Language Processing (NLP) task, we thought it would be useful to look into Zipf's law (see Equation 1). It describes the linguistic phenomenon, where the most frequently occurring word in a language is used twice as often as the second most used word. This can be seen to occur in any kind of text where you can implement a frequency ranking of all the words in the text.(Corral, Boleda, & Ferrer-i Cancho, n.d.) Zipf's law is most often seen in the form of the word frequency function of the infamous distribution figure of the function (see Figure 1).

$$n(r) \propto 1/r^{\alpha} \tag{1}$$

The law states that the frequency n of the r-th most frequent word of a text follows where α is a constant and $\propto$ is the symbol of proportionality. (Corral et al., n.d.)

After taking into consideration Zipf's law we decided to see the word frequency ranking of our data and specifically the words in the titles. You can see the distribution of the words in Figure 1 above. From the distribution shown in the figure, we deducted that our data follows Zipf's law and therefore we concluded that the most frequent words and their distributions would be the same across both classes. Therefore we would not have to be concerned with them affecting the classification process.

Moreover, since the data that was used came from an American publisher (i.e. IEEE Transactions on Computational Social Systems), most of the news titles were from American news headlines. Meaning that our model will be centered around fake news titles in the United States and might not be applicable to identify fake news titles from other places in the world. Furthermore, the language that was used in the data set was also very specific to a certain time period. This is seen in common occurrences of the word "Trump", "Obama", "Russia", etc. All in all, the subjectivity of the data might affect how well the model works in a wider context, but for this research project, we will limit our model to this data.

3 Methods

3.1 Pre-processing

3.1.1 Bag of words

A part of the pre-processing we needed to decide on was the way we are going to present the text as vectors. Considering that we are doing a classifier we decided to use one of the most popular approaches for feature extraction in NLP, which is the bag-of-words (BoW) model. The basics of it consists of two parts, the dictionary of the known words to the model and the way the words are measured in the text, where positioning and structure are ignored. This approach is very flexible as those two aspects can be modified in different ways depending on your goal. This model is text content base. We think that the bag-of-words is a good approach because we wanted to base our classification on word frequency and Zipf's law, which would be our main measure for the text. The BoW approach gives us the opportunity to construct our input vectors based on the frequency of the words in the training data. We took into consideration that our dataset is big enough for us to have a big enough dictionary of known words/tokens. (Brownlee, n.d.)

3.1.2 One-hot approach

The first way we tried to implement the BoW model is by making one-hot encoded vectors of the titles. First, we represented each individual title as its own binary vector representation of 0s and 1s, where the positions of the 1s in the vector indicated the words from the vocabulary present in the text. We also made our training output vectors to be one-hot representations of the two classes fake or real. However, when we tried to encode the text in this way we ran into a problem where the vectors were too big. Making the vectors this way created vectors of size 29896 because to be able to represent all of the possible words in the text, the vectors were made the same size as the vocabulary. This was a problem as it made the processing of the data and the vectors very long and complicated, therefore it was hard to work with them and use them for the model. That is why we decided to change our method of vectoring our data.

3.2 Model Architecture

3.2.1 Text Vectorization layer

After that, we came to find a preprocessing layer that is used for natural language processing and would help us map the text features of our dataset onto vectors. That is why at the end, we inputted the data into the built-in pre-processing layer from tensor flow called the text vectorization layer. The layer first strips the strings from punctuation and makes them lower case, then it tokenizes each string and transforms it into a vector of length 100 to use in the next layer of the model. In order to transform the strings into vectors, the text vectorization layer associates each word with an arbitrary number and uses the assigned number of each word to transform each title into numerical vectors. This then results in a vector where the first position in the vector is the integer representation of the first word of the title and similarly for all other positions, should the input be smaller than one hundred words then the corresponding positions are zero and if the input includes a word that does not have a numerical representation, then the word at that position gets assigned to 1, which indicates an unknown word. Moreover, before inputting the data into the layer we also use two other methods for pre-processing. We transform the data into tensors and afterwards, we adapt the data into a vocabulary and batch it, so that it creates the vocabulary faster. (Team, n.d.)

3.2.2 Embedding layer

The next layer of the model deals with embedding the vectors from the previous layer. This means that each index is associated with a vector of size 32. This layer can be understood as a look-up table, that is able to learn the output vectors during the training of the model. This means that unlike the input the output will indicate some kind of relationship, with vectors that are close to each other being, for this task, similar to each other.

3.2.3 Pooling Layer

The output of the embedding layer is then used in a 1D average pooling layer, this layer reduces the dimensions of the input, by averaging over them. This means that the input shape (samples, num_words, 32) will be transformed into the output shape (samples, 32). This allows for input of fixed shape for the next layer without having to pad the sentences so that they are of equal length. This vector can now be understood as a representation of the sentence.

3.2.4 Hidden Layers

The next three layers are fully connected layers of 32 neurons each, with the RELU activation function, which is zero for values less or equal to zero and otherwise passes on the positive value that is received. The RELU activation function was chosen as it introduces non-linearity, while still being computationally inexpensive.

3.2.5 Output Layer

The output layer consists of two neurons, which represent the two categories namely if the text is fake news or if it is not. The activation function of this layer is the softmax function (see Equation 2), which transforms the output of the network into a probability distribution and normalizes it by the summation of all the outputs, meaning that each neuron outputs a value between 0 and 1. This means that the model will output a vector of length 2, with the probabilities of a certain text being fake news and real news.

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{L} e^{x_j}} \tag{2}$$

With L being the number of neurons in the layer and x_i being the sum of all inputs of the i-th neuron in the layer.

3.3 Cross-Entropy Loss

As for the loss function, the model uses the cross-entropy loss function which simply compares the predicted probability distribution from the model against the expected probability (see Equation 3). This loss function takes the output values from the model and compares them to the actual outputs from the dataset and calculates the ”distance” between the two vectors. Since this loss uses a logarithmic function, it gives large loss values for differences close to 1, and smaller loss values for differences close to 0 making it a perfect loss function for our classification model (Koech, 2020).

In Equation 3, K are all classification classes, which in our case are fake news and real news, $p(x)$ is the true probability of this class being the correct output and is the label of the input encoded using 1-hot-encoding. $q(x)$ is the probability of the predicted class, which can be gathered from the output of the last layer of the neural network that uses the softmax function to output the predicted probabilities of each class. $H(p,q)$ is the loss which is then used by the optimizer to adapt the weights of the network.

$$H(p,q) = -\sum_{x \in K} p(x) log(q(x)) \tag{3}$$

3.4 Adam Optimizer

The model uses the Adam optimizer (Kingma & Ba, 2014), to optimize the weights of the network, we chose this optimizer, as this optimizer is efficient in both memory usage as well as computation speed, furthermore, it is a commonly used optimizer for NLP tasks. This optimizer is based on the adaptive gradient algorithm as well as the root mean squared propagation algorithm both use individual learning rates for each network weight in the model. This optimizer first calculates the gradients (g_t) of the loss function (f) given the previous state of the network (θ_{t-1}) by using partial derivation, as can be seen in Equation 4.

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{4}$$

The gradient (g_t) is then used in combination with the previous first (m_{t-1}) and second (v_{t-1}) estimate of the first and second moment to estimate the new first (m_t) and second (v_t) moment of the function (see Equation 5). β_1 and β_2 are the decay rate of the first and second moment respectively and can be set by the user to the desired value. They are therefore restricted between 0 and 1 with the lower bound being inclusive and the upper bound being exclusive. Usually, values close to 1 should be chosen normally.

$$\begin{aligned} m_t &= \beta_1 * m_{t-1} + (1-\beta_1) * g_t \\ v_t &= \beta_2 * v_{t-1} + (1-\beta_2) * g_t^2 \end{aligned} \tag{5}$$

As the estimate of the first and second moments are based on the previous values, they need to be initialized before this optimizer can be used. They are initialized to a null vector, which means that both moments are biased towards null. In order to counteract this, further calculations (see Equation 6) are used to correct this bias ($\hat{m}_t$ and $\hat{v}_t$).

$$\begin{aligned} \hat{m}_t &= m_t/(1-\beta_1) \\ \hat{v}_t &= v_t/(1-\beta_2) \end{aligned} \tag{6}$$

These bias-corrected moments are then used to calculate the new parameters of the network (see Equation 7). α is the limit of the step size for the learning rate, as long as $|\hat{m}_t/(\sqrt{\hat{v}_t}+\varepsilon)| \leq 1$, which is always the case with the exception of very sparse cases, where the gradient was zero at all previous timesteps. ε is a term that is used to prevent a division during the calculation of the step size, it is therefore often set to a small value.

$$\theta_t = \theta_{t-1} - \alpha * \hat{m}_t/(\sqrt{\hat{v}_t}+\varepsilon) \tag{7}$$

For our network, we decided to stick with the standard values of the toolbox (Keras) that we used, this means that α has a value of 0.001, β_1 has a value of 0.9, β_2 has a value of 0.999 and ε has a value of $1e-07$.

3.5 Model Regularization

In order to prevent the model from overfitting, we decided to use the L2 regularizer (see Equation 8), which penalizes higher weights, as these usually correspond to a more flexible model. θ in the equation corresponds to all weights of the network and l is the factor by how much this regularization factor should influence the loss.

$$reg(\theta) = l * \sum_{w \in \theta} w^2 \tag{8}$$

With Keras, it is possible to set the regularizer for each layer separately, though we decided to give all layers the same regularizer with the same value for l. Only the pooling and text vectorization layer do not have this regularizer, since there are no weights to which this regularizer can be applied.

In order to determine an appropriate value for l, we decided to use k-fold cross-validation with a k of 5 for different values of l in order to determine an appropriate value. The model in each fold was trained for 20 epochs, as through prior testing this amount of epochs seemed to be the point when no major improvements in the accuracy of the training data were seen. In Figure 2 we can see that as expected the loss for the training data goes down as the model gets more flexible and can adapt better to the training data. As for the validation data, this trend is also observed but only until a regularization factor l of 0.0001, after which the loss increases, which indicate that the model is overfitting. Therefore a regularization factor of 0.0001 is the best factor that we found, though it might not be the most optimal factor, as the time constraints only allow for 10 values to be tested.

3.6 K-Fold Cross-validation

In order to further test the robustness of the model, we also used the k-folds cross-validation method. Using this method, we simply divide the data into "k" batches and train the model k times, wherein each training iteration k-1 batches of data is used for training the model, while 1 batch is used to evaluate the performance of the model. Furthermore, the batch of data that is used to test the performance of the model changes with each fold, which allows for a good idea of how well the model generalizes.

4 Results

Our model performs relatively well for the amount of time that we had to work on it. As can be seen in Figure 3 the accuracy of the predictions from the training data increases over subsequent epochs, while the predictions from the testing data fluctuate. However, these fluctuations stabilize after about 7-8 epochs with a final accuracy value of about 91.27% after 20 epochs.

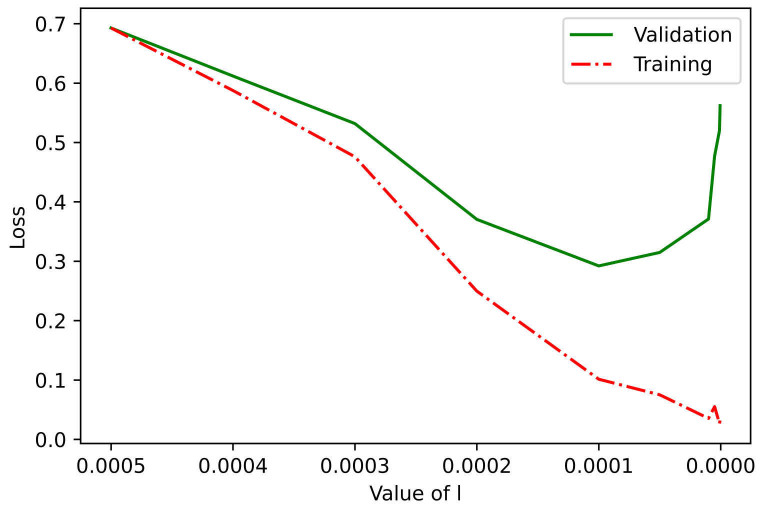

Figure 2: The mean loss over 5 folds given the regularization factor l for both train and validation data.

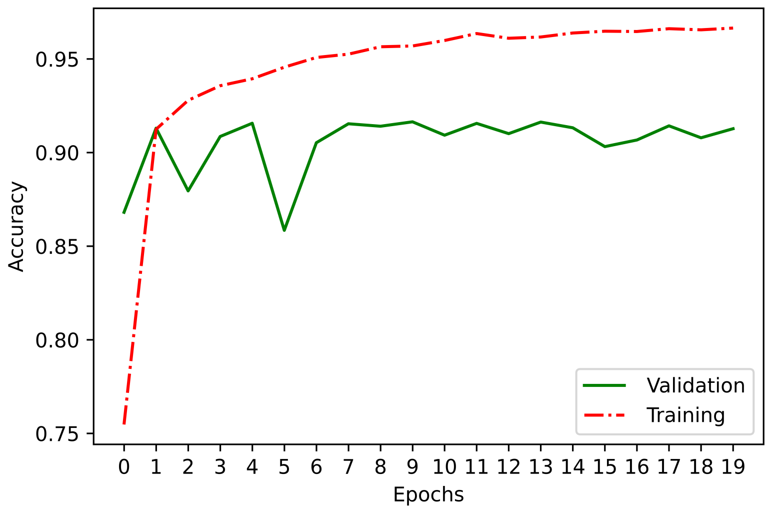

Figure 3: Shows the accuracy of the model after each epoch for the training and validation data.

4.1 Evaluation on article body

It is observed that when we test out the trained model with news article texts instead of news titles, the accuracy of the predictions drops to 65.50% which is only slightly better than flipping a coin to determine if a given article is fake news. This is probably due to the fact that entire news texts have a larger amount of words as well as different words when compared to news titles, and since the model is not equipped to deal with that kind of range and volume, then the accuracy decreases.

4.2 Comparison to other Models

Before we started to work with the data, we were able to explore some models on Kaggle that also aimed to classify news titles as well as texts. Since we used some of them as a reference point for our methods, we will compare our model's performance to the sample model.

The sample model found on Kaggle (TheYazilimci, 2022) has an 88% accuracy level while ours has an average of 91% after k-fold cross-validation. The sample model was trained for 30 epochs while ours was only trained for 20, yet ours still performed better. As seen in Figure 4, the model has a similar accuracy pattern in which it drastically spikes up and then stabilizes as the epochs go on. However, it is seen that the sample model's validation accuracy continues to slowly increase while our model's validation accuracy stabilizes at around 91%. One of the major differences between the two is that the sample model uses the binary cross-entropy loss function, while we use the standard cross-entropy loss, as we have transformed the labels using one-hot encoding. This means that we have two output neurons, while the sample model has only one output neuron.

Although both models are quite similar, we have certain differences in optimization and pre-processing methods where we use the text vectorization layer from Keras while the sample model uses the tokenizer. Moreover, the sample has two dense layers which use different activation functions (RELU and Sigmoid) while our dense layers use the activation functions RELU and softmax.

We think that the performance difference between the models might come from the different pre-processing methods. Although both models use the bag of words approach, we had some differences in the parameters of the pre-processing layer which might have resulted in the accuracy difference because NLP models largely depend on how we transform the text data into numerical vectors.

Another possible reason for the improved performance of our model is that our model uses more layers as well as more neurons in the layers, which allows for a larger search space to find an approximately optimal set of parameters, this means that unless our model overfits it should always find a set of parameters that is at least as good as the smaller model.

5 Discussion

5.1 Conclusion

In this project, we used a feedforward neural network to try to classify news articles as real or fake. Nowadays, algorithms like this can play a crucial role in news websites to help people understand better what to believe. However, designing such a neural network turns out to be not an easy task.

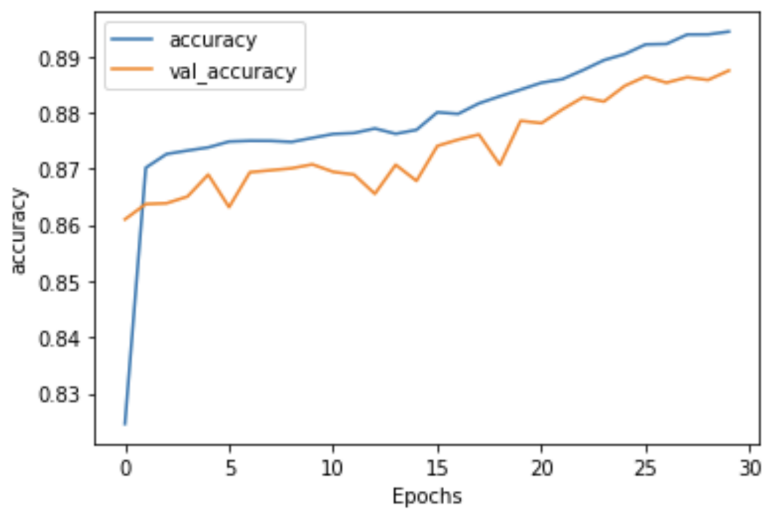

Figure 4: Shows the accuracy of the sample model after each epoch for the training and validation data.

We decided to use the bag-of-words approach, which, however, does not give the best results when dealing with language tasks. Nevertheless, we can see significant accuracy from our K-fold cross-validation - around 90%. However, when training the model only with news titles and testing it against the whole news articles, the accuracy significantly drops - around 65%. There are a lot more words used in the articles (also, words unknown to the model), which results in the model 'struggling' to classify accurately the articles as fake vs. real.

We discussed above our particular design choices as well as the optimizer, loss, and regularization for the architecture.

5.2 Reflection

Due to the nature of NLP models that often use large non-numeric datasets, our biggest feat in this research was deciding how to transform our data into numerical values to feed to the network. We conducted lots of research and decided that the most feasible yet still appropriate level of processing was to feed the data into keras's built-in text vectorization layer. Moreover, we also had to read and research various optimization tools that are appropriate for text-based data and binary classification models.

Another thing that I wished we would have looked into and applied is some kind of early stopping so that the model does not continue training on the data without any improvements in the accuracy of the predictions on the validation set. If we had implemented the early stopping then we could have saved a significant amount of time while training the network, which would have allowed us to maybe test more things.

Furthermore, we should have used the ragged setting for the text vectorization layer, as it allows for a variable length of words in the title instead of a fixed one with padding zeroes if needed. Which would be better suited for a model for which the length of the title is unknown, and it is likely one of the reasons why the model performed poorly using the text as an input. We simply forgot to set this setting to true initially, which means that by the time we noticed it we had already trained the model and fixing this was not an efficient use of time, as the whole model would have had to be retrained again.

5.3 Improvements and potential future research

With only a few weeks to work on this project, we can think of a lot of details and architecture design choices that we could improve. Firstly, the most crucial one is to not use the bag-of-words approach. This method is insufficient to catch deeper meaning in language, as it does not consider the position of the words in relation to each other which means that the intricacies of the sentence can not be determined. As an example let us take the headline "Illinois Governor's Race Shows G.O.P.'s Lurch to Right, Helped by Left" from the New York Times, which if some words are switched such as: "Illinois Governor's Race Shows G.O.P.'s Lurch to Left, Helped by Right" will have a completely different meaning, but this different meaning is not apparent to any model that uses the bag of words approach, as the same words are present. Therefore a model that takes the order of the words into account might be more appropriate, which means that some kind of dynamical system can possibly outperform our current model.

A popular choice nowadays for a dynamical system in natural language processing tasks are Recurrent Neural Networks (RNNs), more specifically Long Short-Term Memory (LSTM) networks. These networks do not use activation function, have memory cells, and most importantly, they do not encounter the vanishing gradient problem. Having memory is what makes this type of network more appropriate and accurate than a feedforward network in natural language processing.

Secondly, there could be an improvement in how we create the word embeddings. When training the model only with titles, we cannot capture that many words, as titles are shorter and concise, and they consist of fewer words. Therefore, there are a lot of out-of-vocabulary words, especially if we test our model with the articles. That could be a reason why the accuracy drops significantly. Therefore, instead of using the embedding layer in our architecture, we could use pre-trained word embeddings. These embeddings are trained on a large data set and all learnt word embeddings can be used in this new task. This would decrease the out-of-vocabulary words and would potentially increase the vocabulary. Another option is at least training the model on the full articles. However, this could have the same or better accuracy only with a sufficient amount of articles and words in them.

Last but not least we definitely could improve how we deal with the data. Currently, there are some articles that do not have a title. We did not remove those and the model is still 'learning' from them. However, they are not showing in any way whether the news is fake or real, as there are both real and fake news without titles. Therefore, it would improve the accuracy of the classification if we simply remove these values.

Apart from improvements to our model, we can look at further potential future research connected to this topic and gathering of the data for new models. According to Horne and Adali (2017), there are certain pointers that indicate whether an article is fake or real. In other words, fake news are not written exactly in a way to look like real, but there is a specific style that is different from the style of real news. This is shown the strongest in the titles of fake news, therefore testing should still mainly remain for the titles. According to the article, indications include fewer punctuation, quotes, fewer nouns and more adverbs. Thus, for future models, it would be important to find a way to consider the overall structure of the sentence - punctuation and functions of the words in a sentence, instead of only the words. Then the model will have a deeper insight into the titles.

Furthermore, other types of data around the article might be useful when detecting fake news. Such data can for example be the date of release and the country of release. Furthermore, it is important to have not such centralized training data as we did for our project. Articles from numerous countries and continents have to be included.

References

Brownlee, J. (n.d.). *Deep learning for natural language processing: Develop deep learning models for natural language in python*. Retrieved from `https://machinelearningmastery.com/machine-learning-with-python/`

Corral, , Boleda, G., & Ferrer-i Cancho, R. (n.d.). *Zipf's law for word frequencies: Word forms versus lemmas in long texts.* Public Library of Science. Retrieved from `https://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0129031`

Horne, B., & Adali, S. (2017). This just in: Fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. Retrieved from `https://www.aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/view/15772/14898`

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv*. Retrieved from `https://doi.org/10.48550/arXiv.1412.6980`

Koech, K. E. (2020, October). *Cross-entropy loss function.* Towards Data Science. Retrieved from `https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e#:~:text=than20in202.-,Cross2DEntropy20Loss20Function,from20the20actual20expected20value.`

Lestrade, S. (2017). Unzipping zipf's law. *PLOS ONE*, *12*(8). doi: 10.1371/journal.pone.0181987

Shahane, S. (2021, April). *Fake news classification.* Kaggle. Retrieved from `https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification`

Team, K. (n.d.). *Keras documentation: Textvectorization layer.* Retrieved from `https://keras.io/api/layers/preprocessing_layers/text/text_vectorization/`

TheYazilimci. (2022, May). *Fake news classification 88% acc.* Kaggle. Retrieved from `https://www.kaggle.com/code/theyazilimci/fake-news-classification-88-acc`