CS 7643 Project Report: Bot Busters

Pearl Ranchal Anuj Singh Yuet Wing Frank Wan Jesse Ward
Georgia Institute of Technology
{ranchal, anuj.singh, ywan33, jward94}@gatech.edu

Abstract

Fraudulent and computer-generated content is a major problem for websites that rely on user contributions. This report investigates the use of recurrent and attention-based neural networks to identify fake text. Datasets evaluated include computer-generated reviews, Yelp spam reviews, fake news articles, and Amazon reviews written by non-verified purchasers. We find that performance of our best networks within their problem domain is comparable to state-of-the-art methods with similar data processing pipelines. However, performance was found not to transfer between problem domains. Systematic transfer learning studies suggest that this poor transferrability is likely due to task-specific feature representations at the intermediate layers. Finally, we explore a method for attributing network evaluation scores to particular input words.

1. Introduction / Background / Motivation

The modern Internet relies on user-generated content, including posts on social media websites like Twitter and Facebook and online reviews on sites like Amazon and Yelp. While this allows for greater interactivity and a more open exchange of ideas, unmoderated content also runs the risk of spreading disinformation, such as fake news and fraudulent reviews. The scope of the problem is difficult to determine, but tellingly, in 2020, Fakespot, a fraudulent review detection service, analyzed 720 million Amazon reviews and concluded that about 42 percent of the reviews were not genuine [2]. With the advent of easy-to-use large language models like ChatGPT, the problem of fake user-generated content is expected to only get worse.

Motivated by this, our team investigated the application of several neural network architectures to the problem of fake text detection. Several groups have carried out similar studies on a variety of datasets [12]. These efforts fall into two categories–studies which focus on the text itself, and studies which combine text and other metadata. Here, we focus on classification using text alone. This is because we wished to produce general-purpose tools, and metadata is site-specific and not always available. A major focus of our study will be cross-domain applicability–how well a network trained on one dataset performs on another dataset. Another key focus will be explainability–which features of a text most contribute to a network's classification.

Achieving these goals would result in a tool that is both general-purpose and interpretable. Our networks could potentially be used as a stand-alone system, or incorporated into a larger framework that uses site-specific metadata. The explainability studies would aid human content moderators in reviewing flagged material.

Datasets used in this study include:

- **Fake Reviews:** This dataset was generated by Salminen *et al*. [16] for evaluating classifiers of real vs. machine-generated text. It consists of 20,000 Amazon reviews written by humans and 20,000 fake reviews generated by GPT-2.
- **Yelp:** This dataset, containing reviews of various businesses, was made available by Yelp in 2015 for their Dataset Challenge [5]. The full dataset has more than one million entries; this work uses a subset of the full dataset [10] and has ∼600,000. This dataset has a field indicating whether a review has been flagged by Yelp's proprietary spam detection system, which we use as our target. This dataset is unbalanced, with 86% normal reviews and 14% spam reviews.
- **Fake News:** Fake news is a separate though similar problem domain as fake reviews. This dataset is available on Kaggle [13] and contains 4,636 real and fake news articles, balanced by category.
- **Amazon:** This dataset was scraped from Amazon's website by data scientist Dinda Calista [4] and consists of 7,652 review texts with metadata including whether the review came from a verified purchaser, which we use as our target. The question of whether a review was written by a verified purchaser is similar but separate to the question of whether a review is fraudulent, as a review for a product not purchased through Amazon may still be legitimate. Since reviews from verified vs.

Dataset	Num. Samples	Ave. Len.	Std. Dev.
Amazon	7,652	22.1	22.5
Yelp	608,246	115.5	106.4
Fake Reviews	40,432	67.0	68.9
Fake News	4,636	4631.4	4193.7

Table 1. Statistics on each dataset.

non-verified purchasers may not show different feature distributions, this dataset is expected to be challenging. This dataset is also unbalanced, with 73% of the reviews from verified purchasers and 27% from non-verified purchasers.

2. Approach

Our approach was to evaluate the performance of several different architectures discussed in CS 7643. This includes recurrent neural networks (RNNs) with and without attention, long short-term memory (LSTM) networks with and without attention, and a transformer encoder with self-attention. These models have inductive biases that are useful for analyzing sequence data such as text strings. We also fine-tuned a pre-trained BERT model to serve as a baseline. In all cases, inputs to the networks were word tokens passed through an embedding layer, and outputs were binary classification scores. RNN, LSTM, and transformer encoder networks were built using the PyTorch library [14], and tokenization was accomplished through the spaCy [9] and NLTK [3] libraries. A pre-trained BertForSequenceClassification model was obtained from HuggingFace [18].

Maximum input lengths were determined by examining statistics on typical text lengths for each dataset. The results are shown in Table 1. A maximum length of 50 words was adequate for the Amazon dataset, and 100 words for the Yelp and Fake Reviews datasets. For the Fake News dataset, each article is relatively long at ∼4,000 words. However, input lengths were capped at 100 words to keep this dataset comparable to others.

As noted above, the Yelp and the Amazon datasets are unbalanced, with normal examples far outnumbering anomalous examples. To prevent bias during training, these datasets were re-balanced by retaining all of the minority examples and subsampling the majority class. After balancing, all datasets were randomly shuffled and divided into training, validation, and test sets in a 80:10:10 ratio.

Hyperparameter tuning was accomplished through grid search, followed by a final tuning using the set of hyperparameters that gave the highest validation accuracy. In all cases, the Adam optimizer was used and cross-entropy loss was the optimization criterion. To prevent overfitting, checkpoints are saved only on epochs where the validation loss reaches a record low. Networks trained on a given dataset were additionally evaluated on the other datasets from different problem domains. While classifiers for fake reviews or fake news are common in the literature, this sort of cross-domain performance check is relatively rare. This approach was expected to be successful, at least for networks trained on the computer-generated Fake Reviews, since we suspected that a subset of the Yelp spam reviews, fake news examples, and non-verified purchaser reviews may have been computer-generated. In this case, a network trained on the Fake Reviews dataset should learn to identify these signatures and flag these in alternate datasets.

One problem anticipated early on was regarding vocabulary definition. While defining the vocabulary based on the entire combined dataset seems reasonable, it poses a problem if only a subset of the total vocabulary is used during training, as the embeddings for words not in the training set would not be updated. This means that some words encountered during evaluation time would inject a random input into the network, degrading performance. This was addressed by defining a unique vocabulary for each dataset based on its training set. At evaluation time, input text is converted to word indices based on this vocabulary, and out-of-distribution words receive the '$\langle unk \rangle$' token.

Initial efforts used the entire input text without excluding stopwords, special characters, etc. While this produced networks with high validation accuracy, an examination of attention weights revealed that emphasis was placed on unreliable features such as line breaks and punctuation. Therefore, the processing pipeline was altered to include stemming, lemmatization, and removal of stopwords, whitespace, and punctuation. This lowered validation accuracy, but we expect that the relevant features will be more generally reliable.

Initial attempts at attribution focused on the SHAP library [11]. However, this library was incompatible with our network and data pipeline. Therefore, a more straightforward approach was taken where network scores are evaluated after systematically replacing individual words with the '$\langle pad \rangle$' token, under the assumption that the largest changes in score would occur when the most relevant words are replaced. This approach is similar to the use of attribution masks in convolutional neural networks [6].

The code and curated data for this project can be found at the following Box link: https://gatech.box.com/s/xtzkj4m7v1nyoedhqul9ji8x9w5kbzvg. Network definitions, tuning code, and attribution code were written by the project team. Some utility and data wrangling functions, such as defining the vocabulary, converting between raw text and word indices, and setting random seeds, were adapted from the CS 7643 Assignment 4 starter code.

3. Experiments and Results

The results of hyperparameter tuning are shown in Table 2, and the best hyperparameters for each network can

Dataset	Network	Test Acc.	Precision	Recall	F1
Amazon	RNN w/out attention	0.65	0.74	0.43	0.54
	RNN w/ attention	**0.73**	0.70	0.76	0.73
	LSTM w/out attention	0.70	0.67	0.76	0.71
	LSTM w/ attention	0.71	0.69	0.73	0.71
	Trans. encoder	0.71	0.70	0.73	0.71
	Pre-Trained BERT	0.65	0.67	0.65	0.64
Fake News	RNN w/out attention	0.78	0.75	0.83	0.79
	RNN w/ attention	0.87	0.85	0.90	0.87
	LSTM w/out attention	0.87	0.89	0.83	0.86
	LSTM w/ attention	0.88	0.86	0.91	0.88
	Trans. encoder	**0.90**	0.87	0.91	0.89
	Pre-Trained BERT	0.76	0.76	0.76	0.76
Fake Reviews	RNN w/out attention	0.80	0.84	0.73	0.78
	RNN w/ attention	0.86	0.85	0.87	0.86
	LSTM w/out attention	0.89	0.89	0.89	0.89
	LSTM w/ attention	0.87	0.86	0.88	0.87
	Trans. encoder	**0.90**	0.90	0.90	0.90
	Pre-Trained BERT	0.71	0.80	0.79	0.78
Yelp	RNN w/out attention	0.68	0.67	0.72	0.69
	RNN w/ attention	0.71	0.70	0.72	0.71
	LSTM w/out attention	0.71	0.70	0.75	0.72
	LSTM w/ attention	**0.72**	0.70	0.78	0.74
	Trans. encoder	**0.72**	0.70	0.75	0.73
	Pre-Trained BERT	0.63	0.64	0.63	0.63

Table 2. Hyperparameter tuning results.

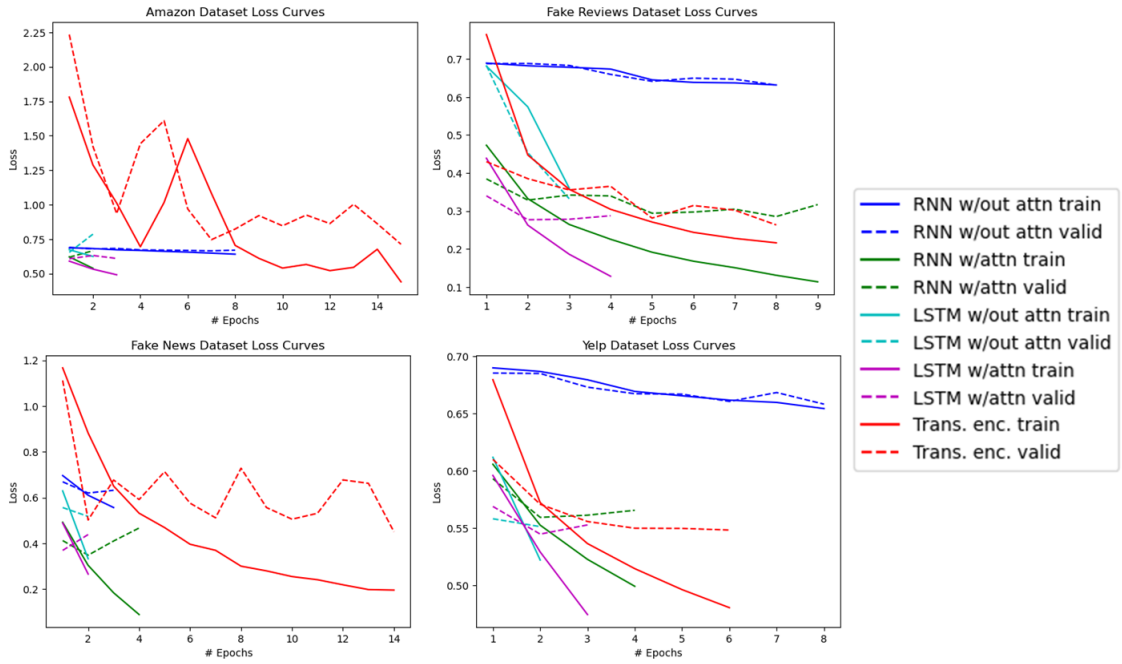

Figure 1. Loss curves for the best networks on each dataset. Smaller models can train in just 2-3 epochs, while the transformer encoder typically takes longer. Checkpoints for subsequent analyses are only saved at epochs with the lowest validation loss.

be found in Appendix A. Loss curves are shown in Figure 1. Our classifiers achieve test accuracies of up to 73% on the Amazon dataet, 90% on the Fake News and Fake Reviews datasets, and 72% on the Yelp dataset. Since user experience would be negatively affected by a classifier with too low a detection threshold, balancing precision and recall was also a high priority for this work. In most cases, precision, recall, and F1 scores are comparable, indicating that our classifiers have low bias.

In all cases, training a classifier from scratch yielded better results than fine-tuning a pre-trained BERT model. Performance improvements from training from scratch vs. fine-tuning were on the order of 8-19% across the board.

Our best performance on the Yelp dataset is comparable to state-of-the-art results on classifiers that also only use text. Refaeli and Hajek [15] achieved a classification accuracy of 73% using a BERT architecture, while Gupta *et al.* [8] achieved 69% accuracy using RoBERTa. Our best test accuracy of 72% is competitive with these results.

Our F1 score of 0.90 for the Fake Reviews dataset is lower than the F1 score achieved by the authors who originally generated this dataset. Specifically, Salminen *et al.* [16] achieved an F1 score of 0.97 using a RoBERTa-based network. However, it should be noted that when the authors performed a variable importance analysis, the most predictive features consisted primarily of stopwords like "at" and "the", and punctuation marks like "!" and ":". These highly predictive tokens are filtered out in our data processing; therefore, a decrease in performance is expected. As mentioned in Section 2 above, initial efforts used the entire unfiltered input text. When this was done, our classifiers achieved F1 scores up to 0.96. Therefore, our results are near the state-of-the-art when a fair comparison is made.

Our best test accuracy of 90% on the Fake News dataset is comparable though somewhat lower than what others have reported. For example, Adiba *et al.* [1] achieve a test accuracy of 92% using a naïve Bayes classifier. This study also employed stemming, lemmatization, and stopword removal, so unlike with the Fake Reviews dataset, any performance differences cannot be due to excluding highly predictive tokens. However, that study did employ a maximum text length of 5000 words, compared to 100 words in this study. A max length of 100 facilitates comparison to other datasets, but limits the features a classifier can train on, so a decrease in accuracy is expected.

While other researchers have built classifiers with Amazon datasets, usually the verified purchaser label is used as a feature, not the target. Therefore, it is difficult to compare performance with others' results. As noted in Section 2, this dataset was expected to be challenging, since "non-verified" does not necessarily mean "fraudulent". As such, feature overlap between the two categories was expected. However, the distributions do not completely overlap, as a test accuracy of 71% shows that there are indicators within the texts of verified vs. non-verified purchasers that allow them to be distinguished at rates greater than one would achieve through random guessing.

Classifier	Amazon Acc.	Yelp Acc.	Fake Rev. Acc.	Fake News Acc.
Amazon	0.71	0.46	0.50	0.45
Yelp	0.44	0.72	0.54	0.50
Fake Reviews	0.47	0.51	0.90	0.54
Fake News	0.53	0.48	0.47	0.90

Table 3. Performance of each transformer encoder classifier on test sets from each dataset.

To check cross-domain transferability, the best transformer encoders from each dataset were evaluated on other datasets' test set. The results are shown in Table 3. In each case, a classifier evaluating a dataset outside of its original problem domain performs no better than a coin flip. This means that features that characterize, *e.g.*, a computer-generated text are different from the features that characterize fake news articles or spam reviews. This was surprising, especially in the case of the Fake Reviews classifier, since it was expected that a subset of the anomalous examples in the other datasets would have been computer-generated. However, there is no evidence that this is the case here. These results underscore the importance of ensuring that the training set is drawn from the same distribution as the examples the classifier would encounter during deployment.

Even if the trained networks are not directly applicable to out-of-distribution datasets, one might still ask whether they have learned useful feature representations that could be used in constructing a classifier in a different problem domain. To check this, we performed a series of transfer learning studies with our transformer encoders. For each problem domain, the optimal architecture was initialized with either random or optimal weights. Next, we froze either the embedding layers alone, or the embeddings plus the transformer encoder layers. The resulting networks were tuned on datasets other than the one the architecture was originally optimized for. The results are shown in Table 4.

The lower layers of a neural network perform feature extraction, while the upper layers perform classification. A linear classifier trained using features from a deep neural network (DNN) with random weights can often still perform better than random guessing, because the DNN generates multiple nonlinear transformations of the input data which may still be useful [7]. Following a procedure adapted from Sandoval-Segura *et al.* [17], we use networks with randomly initialized feature extractors as a baseline for performance. If the classification head achieves better performance when the feature extraction layers are instead loaded with pre-trained weights from an alternate

Frozen Layers	Network Originally Tuned For	Initial Weights	Tuned On Amazon	Yelp	Fake Reviews	Fake News
Embeddings and Transformer Encoder	Amazon	pre-trained	–	0.53	0.57	0.53
		random	–	0.58	0.77	0.70
	Yelp	pre-trained	0.66	–	0.64	0.60
		random	0.68	–	0.73	0.61
	Fake Rev.	pre-trained	0.64	0.56	–	0.55
		random	0.68	0.50	–	0.65
	Fake News	pre-trained	0.60	0.52	0.52	–
		random	0.68	0.54	0.68	–
Embeddings Only	Amazon	pre-trained	–	0.60	0.84	0.76
		random	–	0.58	0.82	0.73
	Yelp	pre-trained	0.67	–	0.81	0.70
		random	0.66	–	0.85	0.74
	Fake Rev.	pre-trained	0.68	0.65	–	0.70
		random	0.68	0.65	–	0.76
	Fake News	pre-trained	0.68	0.66	0.82	–
		random	0.72	0.66	0.84	–

Table 4. Transfer learning validation accuracy, by initialization.

task, the feature extractors of the alternate network can be said to generate useful representations for the new classification task. Comparing the performance on pre-trained vs. random weights in Table 4, we find that when both the embeddings and transformer encoder layers are frozen, surprisingly, the random weights typically perform much better than the weights learned from an alternate classification task–sometimes by a large margin ($\sim$20%). This suggests that features learned by alternate classification tasks are vastly different even though these tasks may have superficial similarities. On the other hand, when only the embedding layers are frozen, final performance using pre-trained vs. random weights is comparable. The embeddings learned for one task are not especially relevant for the other tasks, but they are also not detrimental. This suggests that the task-exclusive feature representations are being produced in the transformer encoder layers rather than the embeddings, and explains the poor generalizability between problem domains shown in Table 3. This is likely also why the pre-trained BERT model did not work as well as the networks trained from scratch in Table 2.

4. Custom Explainer

To understand the model's predictions and gain insights into its decision-making process, we implemented a custom explainer using a masking technique. The goal was to identify the importance of individual tokens in a given input sentence by observing how the model's predictions change when specific tokens are masked.

4.1. Token Masking and Scoring

The custom explainer takes the trained model and a data loader as input. We first set the model to evaluation mode to avoid updating the weights during inference, then extract sentences from the data loader as inputs for analysis. We then input the same sentence multiple times, each time masking the token of interest with the '$\langle pad \rangle$' token while leaving the rest of the sentences unmasked, allowing the contextual information to remain the same. The change in the model's prediction score then correlates with the degree of positive or negative impact of that corresponding word.

Since the raw prediction scores can have varying scales, we applied softmax normalization to the scores. This step ensures that the scores lie in the range [0, 1], making them directly comparable.

4.2. Data Visualization

The custom explainer visualizes token scores using a bar chart. The x-axis represents the positions of the tokens in the input sentence, while the y-axis displays the change in prediction scores. Tokens with positive scores indicate that the presence of that token increases the model's confidence in its assignment. An example is shown in Figure 2.

4.3. Interpreting the Results

By examining the token scores on the bar chart, we can identify the most influential tokens in the model's decision-making process. Tokens with high positive scores are likely to have a significant impact on the model's prediction, while tokens with high negative scores may play a role in ruling

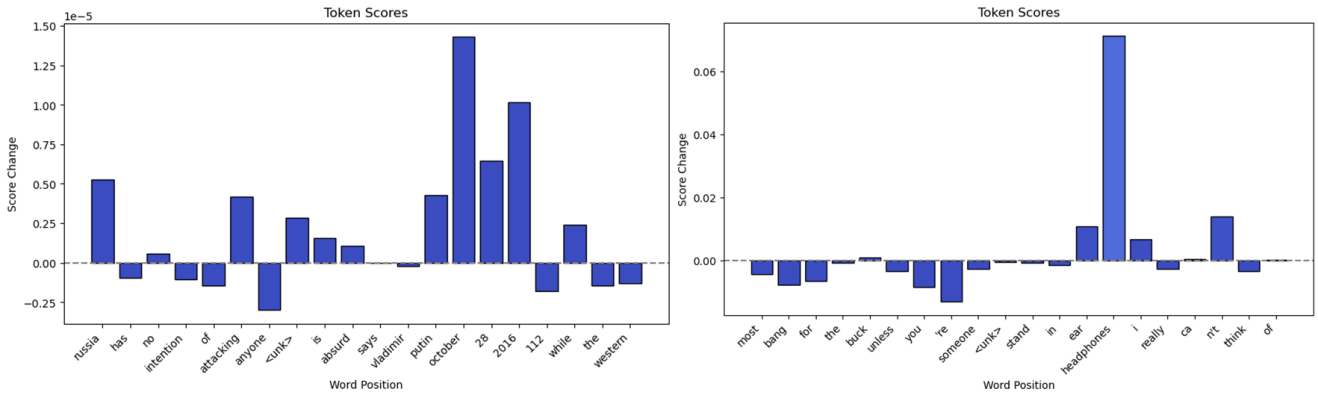

Figure 2. Examples of word explainability score plots. Left: Example from a fake news story. Right: Example from an Amazon review from a non-verified purchaser.

out certain predictions. Figure 2 shows two examples of how this can be useful. On the left is the word scores for a fake news story that a network correctly classified with high confidence. The highest word scores are placed on the article date, October 2016, which makes sense in light of political events at the time. The right shows the word scores from an Amazon review from a non-verified purchaser, where the emphasis has been placed on the word "headphones". While this result is not useful in isolation, a wider examination of the dataset revealed many examples where "headphones" received a high word score, in addition to related words like "sound" and "earbuds". In aggregate, this may indicate that customers are able to find better deals on headphones and earbuds at retailers other than Amazon.

4.4. Difficulties Encountered

The original intent was to implement SHAP's model-agonistic explainers, like its Explainer, KernelExplainer, and DeepExplainer, to provide a thorough understanding of how our model weighted individual words differently. However, this proved to be difficult to achieve. For example, SHAP's Explainer class required a tokenizer that was not readily available for our custom model, the KernelExplainer was not able to parse out tensors within the model, and the DeepExplainer used package versions incompatible with the ones we used to train the model. All these are good lessons learned on the difficulty of cross library compatibility of deep learning models while necessitating the development of a custom explainer to achieve a similar goal.

During the development of our custom explainer, we also encountered issues where our data wrangling process prevented successful translations of the tokens back to the original text, such as text lemmatization and removal of stop words. This required a development of a new model that did not perform such data wrangling, as well as a function that mapped the word indices back to the original text. This yielded more interpretable results.

5. Conclusions and Future Work

We successfully trained a variety of neural networks on similar classification tasks and achieved performance comparable to what has been reported by other researchers. However, despite the superficial similarity between these classification tasks, we found that networks trained for one task did not perform well on the other tasks. After performing a series of transfer learning studies, we determined that these task-exclusive representations are being generated at the intermediate network layers, not the embeddings.

Our custom explainer provides valuable insights into the inner workings of our trained models. It helps us understand which parts of the input sentence are crucial for a model's predictions and offers transparency in the decision-making process. This analysis enhances the interpretability of the model and enables us to make informed conclusions about its performance and behavior.

One future improvement that may boost performance on the highly unbalanced Amazon and Yelp datasets would be to train using a weighted loss function rather than subsampling the majority classes as we did here. Training with, *e.g.*, the class-balanced focal loss function used in CS 7643 Assignment 2 would allow us to still use all of the data without introducing bias into our model.

Another improvement would be to employ some of the semi-supervised techniques discussed in CS 7643 Lesson 18. Specifically, augmenting a labeled dataset for one classifier with high-confidence pseudo-labeled examples from another dataset has been shown to boostrap performance in many other contexts. The Amazon dataset in particular may benefit from this due to its small size.

Student Name	Contributed Aspects	Details
Pearl Ranchal	Data Wrangling and Model Tuning	Generated statistics on average text length per dataset. Trained and tuned vanilla RNN and LSTM with attention. Fine-tuned pre-trained BERT models on each dataset.
Anuj Singh	Data Wrangling and Model Tuning	Incorporated stemming and lemmatization into codebase. Trained and tuned vanilla RNN and LSTM without attention.
Yuet Wing Frank Wan	Model Explainability Studies	Evaluated the use of SHAP framework for our models. Implemented algorithms to determine what portions of the input text are most relevant to model classification.
Jesse Ward	Data Wrangling and Model Tuning	Wrote data cleaning, balancing, and tokenization code. Trained and tuned transformer encoder classifiers. Cross-domain performance and transfer learning studies.

Table 5. Contributions of team members.

6. Work Division

The division of labor between team members is shown in Table 5.

References

[1] Farzana Islam Adiba, Tahmina Islam, M Shamim Kaiser, Mufti Mahmud, and Muhammad Arifur Rahman. Effect of corpora on classification of fake news using naive bayes classifier. *International Journal of Automation, Artificial Intelligence and Machine Learning*, 1(1):80–92, 2020. 4

[2] Julia Bergman. Amazon being targeted by connecticut official for "rampant" fake reviews, Dec 2022. 1

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009. 2

[4] Dinda Calista. Dindatisi/amazon fake reviews detection. 1

[5] Yan Cui. An evaluation of yelp dataset. *arXiv preprint arXiv:1512.06915*, 2015. 1

[6] Ruth Fong and Andrea Vedaldi. Explanations for attributing deep neural network predictions. *Explainable ai: Interpreting, explaining and visualizing deep learning*, pages 149–167, 2019. 2

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 4

[8] Priyanka Gupta, Shriya Gandhi, and Bharathi Raja Chakravarthi. Leveraging transfer learning techniques-bert, roberta, albert and distilbert for fake review detection. In *Proceedings of the 13th Annual Meeting of the Forum for Information Retrieval Evaluation*, pages 75–82, 2021. 4

[9] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python. 2020. 2

[10] Yelp Inc. Yelp dataset, Mar 2022. 1

[11] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. 2

[12] Rami Mohawesh, Shuxiang Xu, Son N Tran, Robert Ollington, Matthew Springer, Yaser Jararweh, and Sumbal Maqsood. Fake reviews detection: A survey. *IEEE Access*, 9:65771–65802, 2021. 1

[13] Nop.ai. Real and fake news dataset., Oct 2019. 1

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 2

[15] David Refaeli and Petr Hajek. Detecting fake online reviews using fine-tuned bert. In *Proceedings of the 2021 5th International Conference on E-Business and Internet*, pages 76–80, 2021. 4

[16] Joni Salminen, Chandrashekhar Kandpal, Ahmed Mohamed Kamel, Soon-gyo Jung, and Bernard J Jansen. Creating and detecting fake reviews of online products. *Journal of Retailing and Consumer Services*, 64:102771, 2022. 1, 4

[17] Pedro Sandoval-Segura, Vasu Singla, Jonas Geiping, Micah Goldblum, and Tom Goldstein. What can we learn from unlearnable datasets? *arXiv preprint arXiv:2305.19254*, 2023. 4

[18] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020. 2

A. Appendix

The best model parameters for each problem domain and architecture are shown in Table 6.

Dataset	Network	Best Hyperparameters
Amazon	RNN w/out attention	batch size = 128, embedding size = 128, hidden size = 128, lr = 1e-4, num epochs = 8
	RNN w/ attention	batch size = 128, embedding size = 128, hidden size = 256, num layers = 3, dropout = 0.1, lr = 1e-3, num epochs = 2
	LSTM w/out attention	batch size = 64, embedding size = 64, hidden size = 64, lr = 1e-3, num epochs = 2
	LSTM w/ attention	batch size = 64, embedding size = 64, hidden size = 64, num layers = 3, dropout = 0.3, lr = 1e-3, num epochs = 3
	Trans. encoder	batch size = 256, embedding size = 256, num heads = 4, num layers = 2, feedforward dim = 8192, dropout = 0.3, lr = 1e-3, num epochs = 15
Fake News	RNN w/out attention	batch size = 32, embedding size = 128, hidden size = 128, lr = 1e-3, num epochs = 4
	RNN w/ attention	batch size = 32, embedding size = 128, hidden size = 128, num layers = 1, dropout = 0.0, lr = 1e-3, num epochs = 4
	LSTM w/out attention	batch size = 64, embedding size = 128, hidden size = 128, lr = 1e-2, num epochs = 2
	LSTM w/ attention	batch size = 32, embedding size = 128, hidden size = 128, num layers = 3, dropout = 0.1, lr = 1e-3, num epochs = 2
	Trans. encoder	batch size = 16, embedding size = 128, num heads = 8, num layers = 2, feedforward dim = 1024, dropout = 0.5, lr = 1e-3, num epochs = 14
Fake Rev.	RNN w/out attention	batch size = 32, embedding size = 128, hidden size = 128, lr = 1e-4, num epochs = 8
	RNN w/ attention	batch size = 64, embedding size = 128, hidden size = 256, num layers = 3, dropout = 0.1, lr = 1e-3, num epochs = 9
	LSTM w/out attention	batch size = 32, embedding size = 128, hidden size = 128, lr = 1e-3, num epochs = 3
	LSTM w/ attention	batch size = 128, embedding size = 128, hidden size = 256, num layers = 3, dropout = 0.3, lr = 1e-3, num epochs = 4
	Trans. encoder	batch size = 64, embedding size = 128, num heads = 2, num layers = 2, feedforward dim = 2048, dropout = 0.3, lr = 1e-3, num epochs = 8
Yelp	RNN w/out attention	batch size = 64, embedding size = 128, hidden size = 128, lr = 1e-4, num epochs = 8
	RNN w/ attention	batch size = 32, embedding size = 128, hidden size = 128, num layers = 1, dropout = 0.0, lr = 1e-3, num epochs = 4
	LSTM w/out attention	batch size = 128, embedding size = 32, hidden size = 32, lr = 1e-2, num epochs = 2
	LSTM w/ attention	batch size = 64, embedding size = 128, hidden size = 256, num layers = 3, dropout = 0.3, lr = 1e-3, num epochs = 3
	Trans. encoder	batch size = 64, embedding size = 64, num heads = 4, num layers = 2, feedforward dim = 1024, dropout = 0.1, lr = 1e-3, num epochs = 6

Table 6. Best hyperparameters for each problem and architecture.