Federated learning on text data

Gabriel Sperrer*
Vienna University of Technology

Maximilian Kleinegger*
Vienna University of Technology

*All authors contributed equally to this research.

1 Introduction

This paper describes our approach to the second exercise in the university course "Security, Privacy, and Explainability in Machine Learning" at the University of Technology, Vienna. We chose topic 3.2.3, "Federated learning on text data (or other sequential data)" and focused on text classification using the Reuters-21587 dataset.

2 Related literature

Text classification, as discussed in this article, is a supervised learning task and has been of interest as a machine learning topic for decades. The main goal is to predict one or multiple labels based on a sequence of words, such as a document. Some of the early approaches relied on the statistical properties of texts and used simple machine learning models, such as linear models or decision trees [7], for classification. Amazingly, some of these approaches, such as Support Vector Machines [9], are still among the best-performing models today [1].

2.1 Text preprocessing

One of the main challenges in text classification is common among most NLP tasks, namely, extracting meaningful features from words that enable understanding of how words are related and what they mean in the context they are used. To be able to apply mathematical operations on words, and for machine learning tasks in particular, they need to be converted to vectors.

A relatively straightforward approach is to ignore the dependencies among words and only rely on the count of the corresponding words in a document. These models are called word n-gram models and represent a purely statistical perspective of language. A simple yet powerful approach to this calculation was already introduced in 1972, called "TF-IDF", a measure of the importance of a word in a document compared to the overall corpus. Equation 1 shows the corresponding calculation, with $tf(t,d)$ being the relative frequency of a term t in a document d and $f_{t,d}$ representing the raw count of a term. When combined with Support Vector Machines for classification, this approach can yield great results in text classification challenges [6]. Due to the simplicity and robustness of these N-gram models, they are still used for some NLP tasks. Today, however, TF-IDF is most commonly used as a preprocessing step for text classification algorithms.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

In recent years, neural networks have become more and more popular in many domains, and their application has outperformed statistical approaches to NLP. However, the vectors resulting from counting words or measuring word frequency are sparse and high-dimensional and therefore don't work as well when training neural networks or deep learning methods. A new approach relying on embedding vectors of fixed size independent of the size of the vocabularity was therefore introduced. The main idea for those vectors was to represent probabilities of a certain word appearing at a certain position [3]. The great thing is that by modeling the algorithm this way, words with similar semantics and meanings will have similar vectors. This means that the machine learning algorithm can learn how similar or different words are, and therefore retain much more information compared to purely statistics methods. This led to the proposal of Word2Vec in 2013, a widely used pre-trained word embedding that can be used for many NLP tasks [11].

2.2 Text classificiation

Once the text is adequately preprocessed, it needs to be classified. The simple, n-gram-based classification algorithms that were already briefly discussed above, have been superseded in recent years by neural networks and deep learning algorithms. One of the main challenges in developing those algorithms specifically for text was the sequential nature and therefore variable length of input and output sequences, a characteristic shared with time-series processing. Recurrent Neural Networks (RNNs), and specifically their successor long short-term memory (LSTM) networks, have been widely successful in solving this issue in both domains [8]. Today, bi-directional LSTMs (bidirectional meaning that a text sequence is processed from front to back and from back to front) are more or less the industry standard in text classification, achieving up to 90% accuracy on prominent datasets [1].

One downside of RNNs is their tendency to ”forget” words that are further from the current input. LSTMs mediate this to a certain extent through their long-term memory and the forget gate. The most recent developments in NLP, however, typically rely on attention mechanisms, which enable a model to learn the importance of other words in the input sequence to a target word [2]. For text and document classification specifically, hierarchical attention networks have been very successful [14].

3 Methodology

This section explains the dataset and ML model used and the approach used to gain the results of this experiment.

3.1 Dataset

We selected the *Reuters-21578* dataset[1] for its longstanding use as a benchmark in text classification and its manageable size, which suited our computational resources. The dataset comprises 10,788 records and includes features such as *text, topics, places, people, orgs, exchanges, date,* and *title.* For our task, we utilized only the *text* and *topics* features as input and target, respectively. The dataset was accessed through the NLTK library [5].

As a side note, we also considered the *AG News*[2] dataset with 10x of *Reuters* dataset size but quickly abandoned it, because training wasn’t feasible with our machines.

3.1.1 Pre-processing

To ensure control over the data, we performed our own preprocessing rather than using pre-processed versions of the Reuters dataset. This involved tokenization, stopword removal, vocabulary construction, and converting tokens into numerical representations. We employed NLTK [5] and scikit-learn [13] for these tasks. Additionally, we applied a maximum length constraint to the text feature to conserve computational resources and added padding to standardize input lengths, using PyTorch [12]. Finally, the target feature was label encoded with scikit-learn [13].

3.2 Model to federate

For the federated model, we aimed for a parallelizable and efficient architecture. We chose a simple, unidirectional LSTM model to maintain simplicity and focus on evaluating the effectiveness of federation rather than optimizing model performance. The architecture consisted of an embedding layer feeding into an LSTM layer, followed by a dropout layer to enhance performance, and a fully connected layer. The hyperparameters are detailed in Table 1, and the model was implemented using PyTorch [12] for flexibility.

[1]`https://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`
[2]`https://paperswithcode.com/dataset/ag-news`

Layer	Parameters
Embedding	**vocab_size**: Vocabulary size, **embed_size**: 128
LSTM	**embed_size**: 128, **hidden_size**: 256, **num_layers**: 2, **dropout**: 0.5
Dropout	**dropout**: 0.5
Fully Connected	**in_features**: hidden_size * 2, **out_features**: output_size

Table 1: Hyperparameters for LSTM Layers

3.3 Baseline

To be able to assess our results and compare the accuracy of our simple model to other models trained on the same dataset mentioned in the literature, we need a baseline. We selected the 2019 paper by Adhikari et al. [1] due to the simplicity of the model and the thoroughness of their analysis. The bi-directional regularized LSTM they proposed achieved an F_1 score of 89.1%.

We implemented a similar centralized model based on a bidirectional LSTM using the same architecture as described in [1]. This allowed us to evaluate our results within the context of our own preprocessing pipeline and provide a direct comparison with the literature.

3.4 Approach

First, we implemented and trained the state-of-the-art algorithm and computed our necessary utility metrics to have a baseline.

Secondly, we implemented the model we wanted to federate and trained it centralized, meaning training one model on all available data. Afterward, we computed the same utility metrics as already mentioned.

Third, we had to partition the data into n different and not intersecting partitions. We have to do this so that we can simulate a real scenario, where every client trains on their data.

Fourth, we implemented a client and server for the federation process. The client trains on the partition of data it receives, and the server then aggregates the weights it receives and redistributes them back to the clients. This is where the federated aspect occurs, and we explored multiple libraries that offer this capability. We decided to move forward with Flower [4], a framework that allows for federating any model, independent of the underlying framework. This was particularly appealing because it did not lock us into using a single machine-learning library. Additionally, Flower has well-written documentation, and the support is very good. Furthermore, Flower offers intuitive and easy-to-use interfaces for both the client and server, making it an excellent fit for our use case.

- Client: When implementing the client, the process begins with partitioning the dataset, as the client's job is to train a local model on its partition of data and return the weights to the server. In this scenario, we did not need to consider vertical federated learning because all clients work with the same set of features, which significantly reduces the complexity of our problem.

- Server: The server acts as the brain of the system, performing the aggregation of the different models' weights. It is crucial to consider an effective strategy for this aggregation. We used the Federated Averaging strategy [10] to average the weights of our neural network.

Lastly, we conducted federated learning across multiple clients via simulation and evaluated the model's performance over different numbers of epochs.

4 Results

In this section, we present the results of our experiments, comparing the performance of centralized and federated models on the Reuters-21587 dataset. We evaluated the models using several utility metrics: accuracy, F_1 score, precision, and recall. The results of these evaluations are summarized in Table 2.

4.1 Centralized model

We first trained and evaluated our baseline centralized models. The bidirectional LSTM model achieved an accuracy of 84.98% and

F_1 score of 85.05%, which is slightly lower than the 89.1% reported in [1], but still indicative of strong performance given our simpler preprocessing pipeline. The unidirectional LSTM model, which we intended to federate, achieved an accuracy of 81.65% and a F_1 score of 81.67%. These results confirm that our centralized models are effective and provide a solid baseline for evaluating the federated approach.

4.2 Federated model

Next, we distributed the dataset across multiple clients to simulate a federated learning environment. We experimented with different numbers of clients to observe the impact on model performance. The results after 100 epochs are detailed in table 2.

Configuration	Accuracy	F_1	Precision	Recall
Centralized bidirectional model	0.849861	0.850516	0.860872	0.849861
Centralized unidirectional model	0.816497	0.816689	0.829657	0.816497
Federated model with 2 Clients	0.798424	0.789575	0.818019	0.798424
Federated model with 5 Clients	0.594995	0.543053	0.804511	0.594995
Federated model with 8 Clients	0.581557	0.532206	0.841511	0.581557
Federated model with 10 Clients	0.523633	0.523633	0.845122	0.523633
Federated model with 15 Clients	0.521779	0.505117	0.868341	0.521779

Table 2: Utility metrics after 100 epochs for different configurations

4.2.1 Federated Learning with 2 Clients

When training with only 2 clients, the federated model achieved an accuracy of 79.84% and an F_1 score of 78.96%. This performance is quite close to that of the centralized unidirectional model, demonstrating that federated learning can be nearly as effective as centralized training with a small number of clients.

4.2.2 Impact of Increasing Clients

As we increased the number of clients, the performance of the federated model decreased. With 5 clients, the accuracy dropped to 59.50% and the F_1 score to 54.31%. This trend continued as we increased the number of clients to 8, 10, and 15, with the model's accuracy declining to 58.16%, 52.36%, and 52.18%, respectively. The corresponding F_1 scores also followed a similar decreasing trend.

The degradation in performance can be likely attributed to the limited size of the Reuters dataset. With only 10.788 records, partitioning the data across many clients results in each client having too few data points to effectively train the model. This leads to poor local model updates, which, when aggregated, result in a less effective global model.

4.3 Analysis of Metrics

The precision of the federated models remained relatively high across different client configurations, indicating that the models were still able to correctly identify a substantial portion of positive instances. However, the recall values dropped significantly as the number of clients increased, suggesting that the models were missing many positive instances. This imbalance between precision and recall is reflected in the declining F_1 scores.

Figure 1 provides a visual representation of the training progress for different configurations, showcasing the accuracy, precision, recall, and F_1 score across 100 epochs. The centralized models, particularly the bidirectional LSTM, show smooth and steady improvement over time. In contrast, the federated models exhibit more fluctuation, especially as the number of clients increases. This instability highlights the challenges of federated learning with limited data.

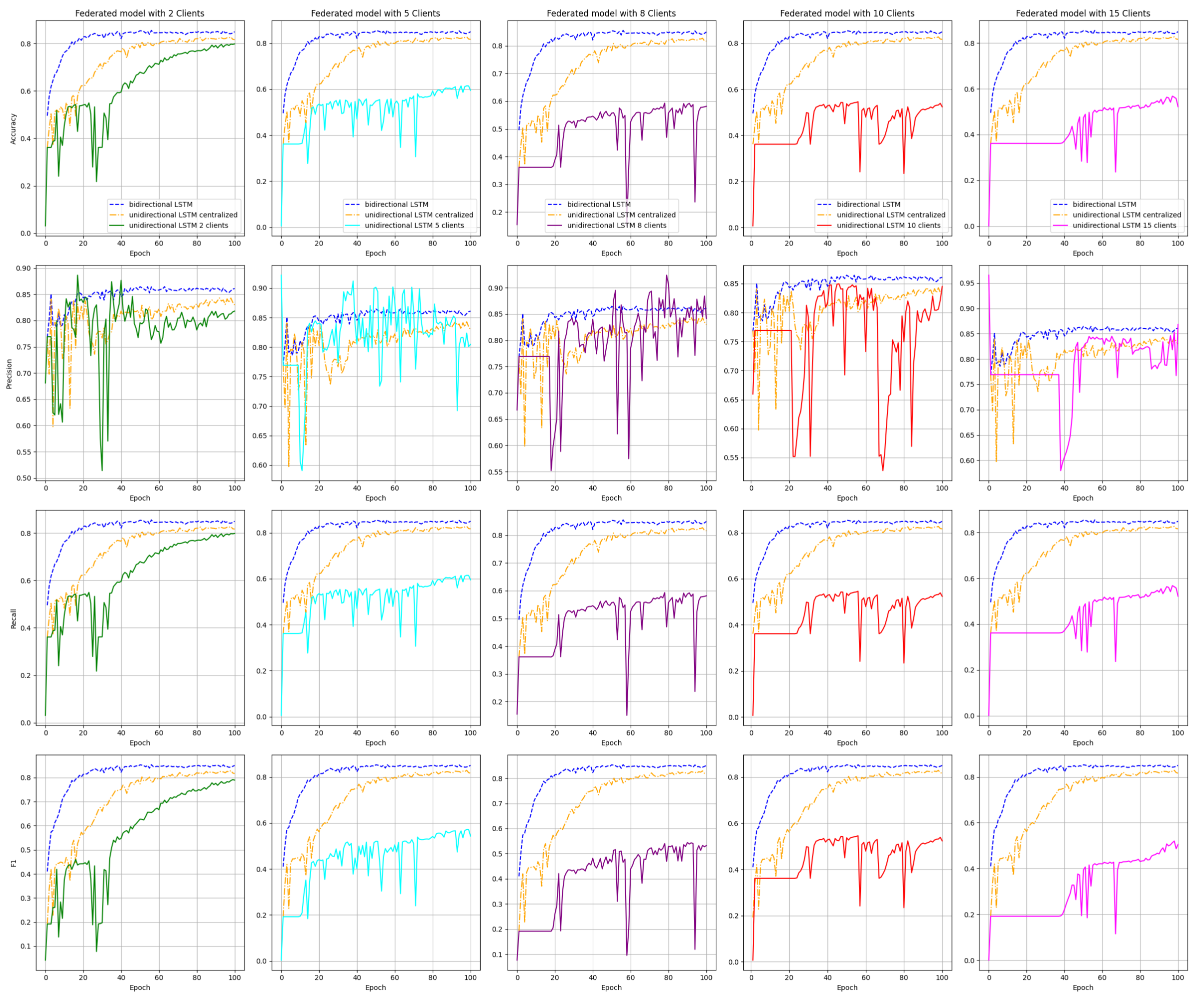

Figure 1: Utility metrics over time for different configurations in comparison to the centralized bidirectional and unidirectional models

4.4 Execution Times

We also measured the execution times for each model configuration. The bidirectional centralized model required 66 minutes to train, while the unidirectional centralized model took 35 minutes. For the federated models, training with 2 clients took 30 minutes, and training with 15 clients reduced the time to 20 minutes. The execution times for other configurations fell within this range. These results indicate that while federated learning can be faster with more clients due to parallel processing, the reduction in training time comes at the cost of decreased model performance.

5 Problems encountered during development

During our project, we encountered several challenges, particularly with the federated learning implementation.

Initially, we used a simple LSTM model and a pre-processed Reuters dataset via TensorFlow, achieving quick and promising results. Motivated by this, we attempted to scale up with the AG News dataset. However, our initial PyTorch LSTM struggled with this larger dataset. To improve performance, we incrementally added an Embedding layer and Dropout layer, which enhanced our centralized model's results.

The federated learning process presented significant hurdles. The server failed to aggregate models due to structural inconsistencies within layers across clients. Switching to a TensorFlow model yielded better-centralized results but did not resolve the federation issues. We also explored TensorFlow Federated but faced prohibitive setup and compatibility challenges.

Ultimately, we reverted to PyTorch and developed a functional model that produced satisfactory results. Additionally, integrating Flower with Jupyter notebooks proved problematic, possibly due to incomplete support for M1 chips in Flower and PyTorch. We could not set up a notebook supporting federated learning and external script imports. After many trials, we consolidated all code into a single notebook, which was the only effective solution.

Moreover, the performance of these models exhibited some randomness. In one training run of a federated model with 10 clients, we achieved unexpectedly high performance with an accuracy of 72.01% and an F_1 score of 69.86%. However, we were unable to replicate these results in subsequent runs. This variability likely stems from the inherent randomness in the training process and the convergence to local optima.

6 Conclusion

In this paper, we explored the application of federated learning to text classification using the *Reuters-21587* dataset. We implemented both centralized and federated models to compare their performance across various configurations. Our baseline centralized models, particularly the bidirectional LSTM, demonstrated strong performance, achieving accuracy and F_1 scores close to those reported in the literature.

Federated learning, while promising to maintain privacy and reduce the need for centralized data storage, presented several challenges. As the number of clients increased, the performance of the federated models declined. This degradation can be attributed to the limited size of the Reuters dataset, resulting in insufficient data for effective local training on each client. Additionally, technical challenges related to model aggregation and framework compatibility further complicate the development process.

Despite these challenges, our federated model with 2 clients achieved results comparable to the centralized unidirectional model, indicating that federated learning can be effective with a small number of clients. However, as the number of clients increased, performance metrics such as accuracy and F_1 score declined significantly, highlighting the need for larger datasets or more sophisticated learning algorithms to improve performance in such settings. When facing similar restrictions to ours, one might also have to reconsider using a simpler model that doesn't require as much data to train. As discussed in our literature review, such simple models relying on statistical measures of word frequency can also generate promising results and require way less data to train.

We also observed that federated learning can reduce training time due to parallel processing, but this benefit comes at the cost of decreased model performance. The execution times of our models showed that while federated learning can be faster, it requires careful consideration of data distribution and model consistency across clients.

In conclusion, our study demonstrates the potential and limitations of federated learning for text classification. Future work should focus on addressing the challenges identified, such as the need for larger datasets and improved model aggregation techniques. Additionally, further exploration into the compatibility and optimization of federated learning frameworks will be crucial in advancing the practical implementation of federated learning in real-word scenarios.

References

[1] Ashutosh Adhikari et al. "Rethinking complex neural network architectures for document classification". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4046–4051.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[3] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model". In: *Advances in neural information processing systems* 13 (2000).

[4] Daniel J Beutel et al. "Flower: A Friendly Federated Learning Research Framework". In: *arXiv preprint arXiv:2007.14390* (2020).

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

[6] Seyyed Mohammad Hossein Dadgar, Mohammad Shirzad Araghi, and Morteza Mastery Farahani. "A novel text mining approach based on TF-IDF and Support Vector Machine for news classification". In: *2016 IEEE International Conference on Engineering and Technology (ICETECH).* IEEE. 2016, pp. 112–116.

[7] Norbert Fuhr et al. "AIR/X-A rule-based multistage indexing system for Iarge subject fields." In: *RIAO.* Vol. 91. Citeseer. 1991, pp. 606–623.

[8] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[9] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *European conference on machine learning.* Springer. 1998, pp. 137–142.

[10] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics.* PMLR. 2017, pp. 1273–1282.

[11] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[12] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[13] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[14] Zichao Yang et al. "Hierarchical attention networks for document classification". In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies.* 2016, pp. 1480–1489.