A Federated Learning approach for text classification using NLP

No Author Given

No Institute Given

Abstract. Text classification is important in many aspects of NLP, such as word semantic categorization, emotion analysis, question answering, and conversation management. Law, health, and marketing are just a few of the professions that have made use of it throughout the last century. We focused on emotion analysis in this research, which includes categories like happiness, sadness, and more. We investigated the precision of three alternative models for assessing the emotional tone of written material. Deep learning models like GRU (Gated Recurrent Unit) and CNN (Convolutional Neural Network) are used in conjunction with the Bi-LSTM (Bidirectional LSTM) model. These three major deep learning architectures have been extensively studied for classification applications. Finally, the model with the greatest accuracy on the dataset was trained using Federated Learning. Using the FL approach, more data may be collected, eliminating data gaps and ensuring data security. The focus of this research is to increase the model's accuracy by collaborating with FL approaches while maintaining data confidentiality.

Keywords: :(NLP · Federated · Deep learning · CNN · GRU · Bi-LSTM · ML)

1 Introduction

Federated learning is a machine learning program where clients(multiple entities) work with each other in solving a problem which is related with machine learning under the cooperation of a central server or service provider. Each client's raw data is stored locally and it is not exchanged or transferred to someone else. Here clients used to train their local models with the data that is stored privately in the framework. Usually, the central server monitors every platform for the purpose of gradient aggregation and then it performs global model updating after it finishes collecting gradients from all platforms. After that it distributes the new updated parameters of the model that is shared, to each and every platform for next-round model training. Federated learning is also implemented in different places like Dataset and Experimental Settings, Experimental Results, Influence of Training Data Size, Model Decomposition Strategy, Influence of Overlapped Entity Number and Generalization of federated learning frameworks.

Text Classification is a process that identifies the pre-defined category for text of various lengths. Text Classification can also be used in many NLP applications that includes sentiment analysis, question answering and topic labeling. Usually, Text classification that is traditional its tasks can be classified into four steps and they are: dimension reduction, text preprocessing, classification and evaluation. The deep learning models has already achieved the place of state-of-the-art results in text classification but the actual problem occurs in uploading or sharing text data to improve model performance. It is not always feasible because of the different privacy requirements of the clients. It is basically implemented with the use of an Induction Network (using meta-learning for classification tasks) or a BART model that is pre-trained after customizing it for every node of the dataset. This helps with a solution where the communication among the nodes is recurrent and also decreases the computational costs efficiently. On the other hand, the letter remains bias-free and works effectively also with sparse data.

2 Related Work

Liu and Miller[1] presented the federated pre-training and fine-tuning of the BERT model using clinical data. They are the first ones who used the federated settings for a large transformer model. In a federated manner, they trained the clinical corpus with both pre-training and fine-tuning methods while using multiple silos of data. Even though original BERT model was trained on Books and Wikipedia but they did it on clinical data using MIMIC-III discharge summary using a federated manner and for that they conducted 6 different experiments.

With continually running speech-based models such as wake word detectors, we present a realistic method based on federated learning to solve out-of-domain challenges. For the wake word "Hey Snips," we conduct an empirical evaluation of the federated averaging strategy. Data from real-world environments, such as users' homes, is generally used to train wake word detectors. Precision is significantly more crucial because the model may be utilized at any time. The usage of federated learning (FL) in the context of an embedded wake word detector is investigated. We believe this is the first experiment of its sort on user-specific speech. To increase convergence, the authors suggest a stochastic variance-reduced gradient descent optimization process (SVRG) that uses both local and global per coordinate gradient scaling. We look at how to train a wake word detector utilizing truly decentralized user data in the first portion of this post. Following that, we'll go through the federated optimization approach and how to replace its global averaging with an adapt dative averaging algorithm inspired by Adam. For Hey Snips, the waking word dataset was crowdsourced. Using crowdsourced data, we test the efficacy of the federated averaging approach for the Hey Snips wake phrase. The outcomes of federated optimization are compared to those of a typical setup, such as a centralized mini-batch using data from randomly randomized train set users. Adam delivers a significant

convergence speedup over standard SGD, which is still under 87 percent after 28 epochs after adjusting the learning rate and gradient clipping on the dev set. When compared to adaptive per-parameter averaging, standard global averaging produces poor results even after 400 communication rounds. The results reveal that the effects are constant across a variety of local training circumstances, with very little advantages as the amount of local training rises. [2]

Federated learning employs client devices to train a Neural Network model, which is subsequently delivered back to the machine learning model's owner, the server. The server compiles these models into a single global model and delivers it back. We repeat this until the global model reaches our desired accuracy. Any transaction and model data in FL is stored on the server. The records and model stored on the server will be lost or destroyed if it is damaged or malfunctioned. If this happens, the model will need to be retrained. Flower - a complete FL framework that separates itself from prior platforms by providing additional tools for running large-scale FL experiments and considering fully diverse FL device situations has been utilized. [5]

FedNer [6] is a Framework where the server usually coordinates multiple clients for the purpose of local model updating and global model sharing. Basically, here the clients are different medical platforms, and they used to train their local models with the data that is stored privately. Usually, the central server monitors every platform for the purpose of gradient aggregation and then it performs global model updating after it finishes collecting gradients from all platforms. After that it distributes the new updated parameters of the model that is shared, to each and every platform for next-round model training. FedNer is also implemented in different places like Dataset and Experimental Settings, Experimental Results, Influence of Training Data Size, Model Decomposition Strategy, Influence of Overlapped Entity Number and Generalization of FedNer Framework.

Healthcare access [7] is proving to be a significant difficulty. Through social media, users from all around the world may share their views and ideas. The categorization job is the most important aspect of this research. Text data may come from a variety of places, including emails, chats, site data, customer evaluations, and feedback. The accuracy of the algorithms employed in DL is used to assess the performance for categorizing the text. The process of text classification include categorizing news, assessing the text based on user opinion, and classifying the content. Fast Text is a simple text classification system. The CBOW and doc2vec models are interchangeable. The doc2vec model and the Continuous Bag of Words (CBOW) model are equivalent. For NLP tasks, the standard RNN model failed miserably. Deep learning in combination with natural language processing (NLP) offers great potential for text classification challenges. Inwardly in their work, they have developed LSTM technology for NLIs. Attention is focused on subdividing the difficulties and solving them in-

dependently. You will find that using two or more technique scans together can improve performance and give better results.

3 Proposed Methodology

3.1 Bi-LSTM

Text data is difficult to discern an intended message from since it is unstructured and includes natural-language components. Traditional text classification techniques include dictionary-based and rudimentary machine learning methodologies. The extraordinary capacity of LSTM to extract complex text information is crucial in text classification. Variable-length sequences benefit from the attention mechanism as well as a more even distribution of weight. We employed Word2vec weights that had been pre-trained with a large corpus, guaranteeing that the model was more accurate. The model's accuracy was evaluated using precision, recall, and the F1-score. For natural language processing (NLP) applications, the LSTM model plus a CNN proved to be unexpectedly successful. Medical literature contains some of the most sophisticated material that only specialists in the area can comprehend. As a result, one critical model for comprehending intricate, one-of-a-kind medical structures was developed. The innovative hybrid CNN-LSTM models produced promising results. Salur et al. developed a novel hybrid model that combines several word embeddings with different learning approaches. Dong et al. used LSTM with label embedding to improve text classification accuracy. An increasing number of research have proven that sequence-based sentiment categorization may be achieved using LSTMs. The LSTM's accuracy is further hindered by its inability to distinguish between different relationships between document components. To solve this problem, a 1D CNN has been suggested. Before being input into the model, the text was preprocessed. This includes, among other things, removing whitespace and unneeded words, converting other words to approximations, and minimizing duplicate words. Convolution is in charge of extracting features from the input text. The attention-based Bi-LSTM is the primary classification component in our model. All the timesteps in a convolution channel will be pooled together to get the greatest possible output value. The attention mechanism gives more weight to "happy" than "sick" in order for a sentence like "Though he is ill, he is cheery" to be evaluated as having positive emotion. Text classification has been employed in a wide range of industries, including law, medical, and marketing. In this article, we present a hybrid Bi-LSTM attention text categorization model. The suggested model's accuracy improved with increasing quantities of training data and training epochs. The classification report of the model is given below:

We have also generated the confusion matrix of this model. In order to show predictive analytics such as recall, specificity, accuracy and precision, confusion matrices are utilized. Confusion matrices are helpful for comparing values such as True Positives, False Positives, True Negatives, and False Negatives directly. The matrix is also given below:

	precision	recall	f1-score	support
class 0	0.94	0.83	0.88	314
class 1	0.90	0.78	0.83	270
class 2	0.90	0.94	0.92	687
class 3	0.88	0.67	0.76	160
class 4	0.89	0.97	0.93	626
class 5	0.65	0.84	0.74	88
accuracy			0.89	2145
macro avg	0.86	0.84	0.84	2145
weighted avg	0.89	0.89	0.89	2145

Fig. 1. Classification report

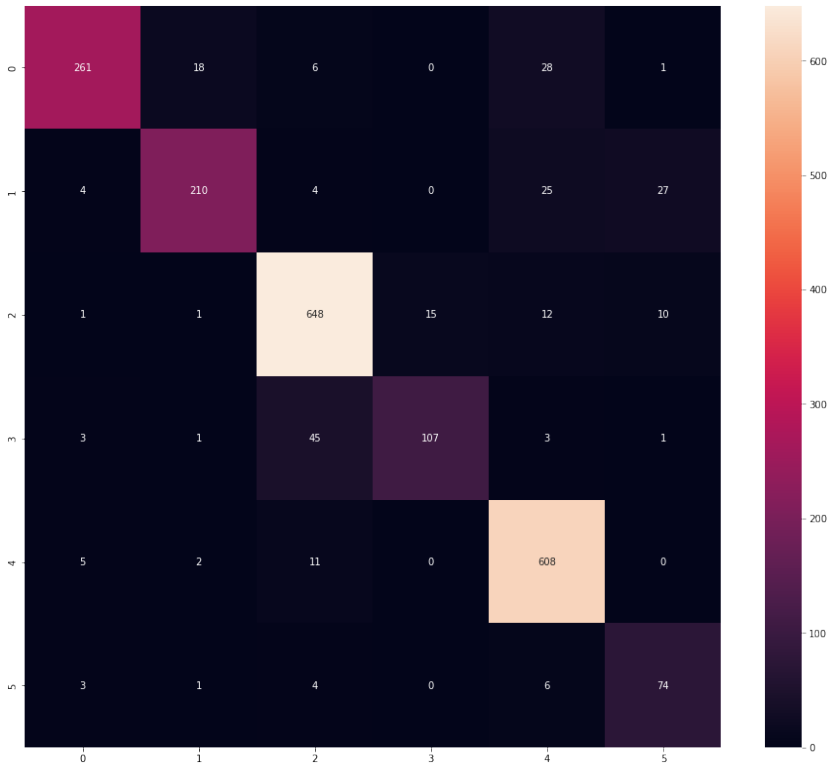

Fig. 2. Confusion Matrix

3.2 GRU

GRUs are a gating mechanism which is used in RNN. In 2014, It was Kyunghyun Cho et al who introduced this model. There is a problem with short-term memory in RNN. When a sequence become large enough, they face difficulty sending information to later time steps from earlier ones. RNNs may leave out essential information at the start when attempting to predict anything from a paragraph of text. During back propagation, the vanishing gradient problem impacts recurrent neural networks. To update a neural network's weights, gradients are used which are actually values. The vanishing gradient problem occurs when a gradient decrease as it propagates backwards in time. If a gradient value goes below a fixed level, for learning it will not be useful after that. In recurrent neural networks, layers that get a minor gradient update cease learning. Those are often the first layers to show up. RNNs may forget what they have watched in longer sequences since these layers do not learn, resulting in a short-term memory. GRUs were created to address the issue of short-term memory. Data flow can be regulated by those built-in devices where are in GRUs. Those are known as gates. In a sequence which data should be retained and which data should be deleted can be determined by these gates. It may then send vital data down a long chain of sequences, allowing it to make predictions as a consequence. This network produces nearly all state-of-the-art recurrent neural network results. Voice recognition system, speech synthesis and text generation all of these use GRUs. This model can be used to create video subtitles as well. In our paper, we have used GRUs for text categorization and achieved 91% accuracy.

	precision	recall	f1-score	support
class 0	0.90	0.87	0.88	314
class 1	0.87	0.87	0.87	270
class 2	0.95	0.90	0.92	687
class 3	0.77	0.94	0.85	160
class 4	0.92	0.96	0.94	626
class 5	0.85	0.73	0.79	88
accuracy			0.90	2145
macro avg	0.88	0.88	0.88	2145
weighted avg	0.91	0.90	0.90	2145

Fig. 3. Classification report

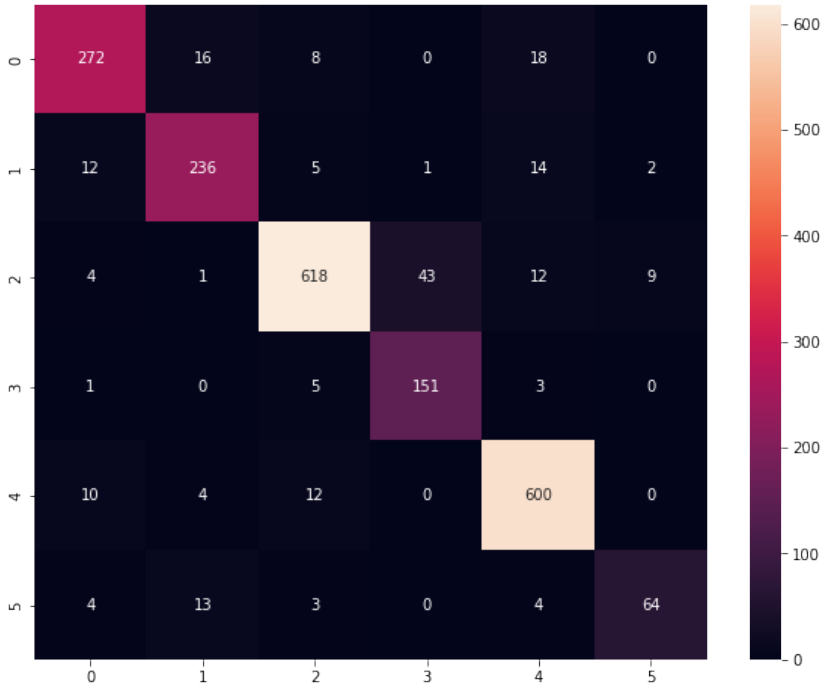

Fig. 4. Confusion Matrix

3.3 CNN

A convolutional layer is a kind of feature map which is further connected with local regions. Each of them takes input according to the kernel and produces an output received from the kernel.

Pooling Layer The pooling layer is mainly used to decrease the parameters and computational complexities of a network. It divides the data into subsections and returns the maximum value from a particular subsection.

Dense Layer Dense Layer is nothing but a method to carry out linear operations in order to convert the input within some predefined weights. It takes an activation function as input.

We have used a CNN model with embedding layer in which the first 2nd layer will be convolutional layers and others are fully connected. The output from the last fully connected layer will be passed into a Sigmoid, ReLu, and Adam which will distribute data amongst three classes. Each layer will be based on 1D feature maps and it will try to extract specific features while holding spatial information.

	precision	recall	f1-score	support
class 0	0.79	0.99	0.88	346
class 1	0.76	0.92	0.83	290
class 2	0.79	0.97	0.87	864
class 3	0.85	0.28	0.42	209
class 4	0.99	0.79	0.88	687
class 5	1.00	0.43	0.60	103
accuracy			0.84	2499
macro avg	0.86	0.73	0.75	2499
weighted avg	0.86	0.84	0.82	2499

Fig. 5. Classification Report

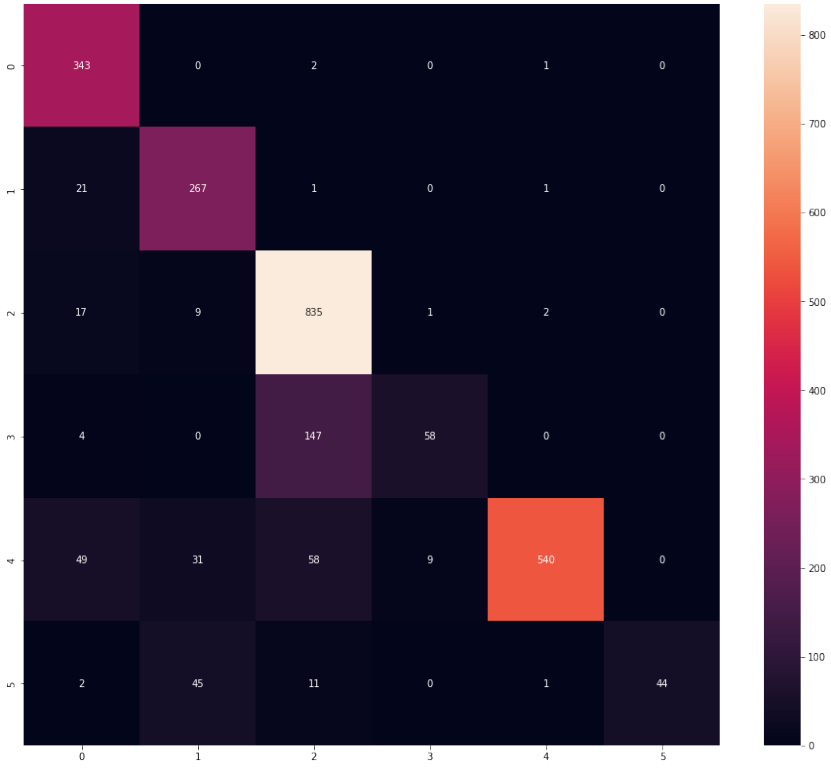

Fig. 6. Confusion Matrix

3.4 Activation Function

An activation function is a crucial factor in machine learning, neural networks or in the field of artificial intelligence. The main job of an activation function is, it basically decides whether a neuron should be activated or not. Moreover, it defines the output of that node given an input or set of inputs.

ReLu It is rectified linear activation function which is basically a piecewise linear function that would only produce output if the input is greater than zero or positive. Otherwise, it will produce the output as zero. Because of its easier training and performing better many neural networks use it as their default activation function.

Sigmoid A sigmoid function is a special form of the logistic function. It has a domain from negative infinity to positive infinity and a range from 0 to 1. This function is monotonically increasing and continuous everywhere. It is an Activision function for neural network.

Adam In training data, we have to update the network weights in a iterative way and for that we use Adam which is an optimization algorithm. It is different from the usual stochastic gradient descent procedure which was usually used before. It is used for accelerating the gradient descent algorithm by considering the exponentially weighted average of the gradients. Using averages make the algorithm converge towards the minima in a faster pace.

Dropout To address overfitting problems, dropout was used after the first dense layer in this model at a ratio of 0.5.

GlobalMaxPooling This is another form of pooling layer. At first, we take the size of the pool equal to the input size so that the output value is the maximum of the entire input.

4 Input Data

4.1 Data-Set

We have a fairly well-labeled dataset that comprises around 21450 unique data points of multi-class emotions. Our dataset contains more than simply sentiment classification; there is more to a sentence's sentiment than just positive, negative, and neutral sentiment. We were trying to figure out what was going on in the mind of the person who wrote the words. Sadness , rage, surprise, joy, love and fear are just a few of the emotions represented in this multi-class

data collection. We chose to create a csv file from all of the text files since arranging the data from a txt file is very time-consuming and difficult. The csv file has 2 columns and they are text and emotions. The Emotions column has a number of different categories, ranging from happy to sadness, to love and fear.

Text	Emotion
i didn't feel humiliated	sadness
i can go from feeling so hopeless to so damned	sadness
i'm grabbing a minute to post i feel greedy wrong	anger
I am ever feeling nostalgic about the fireplace	love
i am feeling grouch	anger

4.2 Data Pre-processing

We have previously processed the data before delving into the model's work. This data processing will allow the models to operate more effectively and with fewer mistakes. To begin, we eliminated the emoji symbols from each sentence in the dataset so that the models could only deal with text. After that, we removed any URLs from the texts. We don't require the URL in our dataset since it doesn't indicate any kind of emotion. Unnecessary punctuation marks may appear in the text. Unnecessary punctuation may lead us to wrong evaluation , so we eliminated all unnecessary punctuation. We divided the data into two sets: the training dataset (which contains about 88 percent of the whole dataset) and the test dataset (which contains just 12 percent of the total dataset). We will use the training dataset to train the models, while the testing dataset will be used to assess the model's accuracy after training. Finally, we ran the dataset through the tokenization algorithm. Tokenization is the process of breaking down a phrase, sentence, or text document into smaller pieces, such as individual words.

5 Experiments and Results

Because it is a relatively new concept, only a few frameworks exist to implement it. TensorFlow has created its own version known as TensorFlow Federated. Also, we utilized Flower for our paper. Flower is a new Federated Learning framework. Unlike TensorFlow Federated and PySyft, which are tied to a specific framework, Flower is designed to operate with all of them. It focuses on providing tools for effectively implementing Federated Learning while allowing you to focus on the training itself. It is really simple to set up a basic Federated configuration in Flower. In addition, the range of compatible devices is extremely broad, ranging from mobile devices to servers and others. As demonstrated in their paper, its design provides for scalability up to 1000s of clients[11]. We have used 100 clients and 10 federated rounds for our simulation. Though we received better accuracy in GRU compared to LSTM for centralized language modeling tasks from our dataset. It showed no difference in performance for federated learning simulation.

We only got better accuracy after increasing the federated rounds. The GRU model has less gates compared to LSTM so it is a little speedier in training but it didn't help for federated settings. The solution for a better accuracy in federated settings could be to increase the dataset and increasing the federated rounds without changing the model architectures.

```
DEBUG flower 2022-05-04 06:37:19,356 | server.py:227 | evaluate_round received 0 results and 25 failures
DEBUG flower 2022-05-04 06:37:19,357 | server.py:269 | fit_round: strategy sampled 50 clients (out of 100)
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5209 - accuracy: 0.3509
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6054 - accuracy: 0.3041
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6210 - accuracy: 0.3275
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6005 - accuracy: 0.3626
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6602 - accuracy: 0.3216
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6045 - accuracy: 0.3099
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5969 - accuracy: 0.3099
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6294 - accuracy: 0.3041
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.5823 - accuracy: 0.3743
(launch_and_fit pid=56989) 6/6 - 4s - loss: 1.5678 - accuracy: 0.3684
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6628 - accuracy: 0.2515
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.4968 - accuracy: 0.3275
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6156 - accuracy: 0.3041
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6261 - accuracy: 0.3041
(launch_and_fit pid=56989) 6/6 - 4s - loss: 1.6097 - accuracy: 0.3626
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.5958 - accuracy: 0.2749
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5529 - accuracy: 0.3743
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6648 - accuracy: 0.2690
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6731 - accuracy: 0.2924
(launch_and_fit pid=56989) 6/6 - 4s - loss: 1.5961 - accuracy: 0.3392
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.6532 - accuracy: 0.3041
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6743 - accuracy: 0.2865
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5626 - accuracy: 0.3509
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.5992 - accuracy: 0.3567
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5754 - accuracy: 0.3392
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.5480 - accuracy: 0.2690
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6209 - accuracy: 0.3626
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.5673 - accuracy: 0.3743
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6442 - accuracy: 0.2690
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.5831 - accuracy: 0.3801
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.5372 - accuracy: 0.3099
(launch_and_fit pid=56989) 6/6 - 4s - loss: 1.5590 - accuracy: 0.3450
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6095 - accuracy: 0.3216
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.5708 - accuracy: 0.3392
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6339 - accuracy: 0.3041
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.6241 - accuracy: 0.3041
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5949 - accuracy: 0.3509
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.5501 - accuracy: 0.3684
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5733 - accuracy: 0.3392
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.7096 - accuracy: 0.2749
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6360 - accuracy: 0.2573
(launch_and_fit pid=56989) 6/6 - 4s - loss: 1.6058 - accuracy: 0.2632
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5835 - accuracy: 0.3567
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.6378 - accuracy: 0.3392
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.6011 - accuracy: 0.3041
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6150 - accuracy: 0.3041
(launch_and_fit pid=56989) 6/6 - 4s - loss: 1.6156 - accuracy: 0.2865
(launch_and_fit pid=56988) 6/6 - 4s - loss: 1.6123 - accuracy: 0.3333
(launch_and_fit pid=56989) 6/6 - 3s - loss: 1.5923 - accuracy: 0.3450
DEBUG flower 2022-05-04 06:38:57,473 | server.py:281 | fit_round received 50 results and 0 failures
DEBUG flower 2022-05-04 06:38:57,621 | server.py:215 | evaluate_round: strategy sampled 25 clients (out of 100)
(launch_and_fit pid=56988) 6/6 - 3s - loss: 1.6299 - accuracy: 0.3216
```

Fig. 7. Accuracy of our model

6 Conclusion

This work discusses detecting emotion using deep learning models and a federated learning architecture. In addition, the top text classification models for usage in the Federated Learning architecture are displayed. A system like this may be used to assess people's emotions and propose movies and television shows. However, the performance of such a model is dependent on a number of aspects, including the quality of data and parameters. Dataset suppliers should also update their databases as soon as new data becomes available in order to augment models through communication loops. If all of these can be maintained, such an architecture can ensure a very high accuracy, similar to or even better than what we showed.

References

1. Liu, D. Miller, T. (2020, February 20). Federated pretraining and fine tuning of Bert using clinical notes from multiple silos. arXiv.org. Retrieved May 5, 2022, from https://arxiv.org/abs/2002.08562
2. Leroy, D., Coucke, A., Lavril, T., Gisselbrecht, T., amp; Dureau, J. (2019, February 18). Federated learning for keyword spotting. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/1810.05512
3. Thakkar, O., Ramaswamy, S., Mathews, R., amp; Beaufays, F. (2020, June 12). Understanding unintended memorization in federated learning. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/2006.07490
4. Stremmel, J., amp; Singh, A. (1970, January 1). Pretraining Federated Text Models for next word prediction. SpringerLink. Retrieved May 6, 2022, from https://link.springer.com/chapter/10.1007/978-3-030-73103-8_34
5. Ge, S., Wu, F., Wu, C., Qi, T., Huang, Y., amp; Xie, X. (2020, March 25). Fedner: Privacy-preserving medical named entity recognition with Federated Learning. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/2003.09288
6. Basaldella, M., Antolli, E., Serra, G., amp; Tasso, C. (1970, January 1). Bidirectional LSTM recurrent neural network for keyphrase extraction. SpringerLink. Retrieved May 6, 2022, from https://link.springer.com/chapter/10.1007/978-3-319-73165-0_18
7. Bhattacharyya, S. (2020, November 17). Author(Multi-class text) classification using bidirectional LSTM amp; Keras. Medium. Retrieved May 6, 2022, from https://medium.com/analytics-vidhya/author-multi-class-text-classification-using-bidirectional-lstm-keras-c9a533a1cc4a
8. Muthukumar, N. (2021). Few-shot learning text classification in Federated Environments. 2021 Smart Technologies, Communication and Robotics (STCR). https://doi.org/10.1109/stcr51658.2021.9588833
9. Stremmel, J., amp; Singh, A. (2020, August 17). Pretraining Federated Text Models for next word prediction. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/2005.04828
10. Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K. H., Parcollet, T., de Gusmão, P. P. B., amp; Lane, N. D. (2022, March 5). Flower: A friendly federated learning research framework. arXiv.org. Retrieved May 6, 2022, from https://arxiv.org/abs/2007.14390