Prize recipient solution documentation guide

This document details the information required to submit your Phase 2 solution materials.

I. **Code submission and result reproducibility**
You package up and send us your code, documentation of dependencies, and any other assets you used. We review your package and make sure that it works and that we can fully reproduce the workflow from raw data to a submission comparable to your best submission. See the Competition prize winner submission template for more guidance on packing up and organizing your code (however, please note that the template's write-up questions differ from the questions for this challenge).

II. **Model documentation and write-up**
You write up answers to our questionnaire, providing important context and documentation so that the beneficiary and the community get the most out of your work.

I. Code submission and result reproducibility

You will need to submit a compressed archive of your code. You don't need to include the raw data we provided, but everything else should be included and organized. If the files are too large to be e-mailed, a Google Drive or Dropbox share (or other comparable method of transferring data) works.

Note: *Please follow these instructions carefully*. The spirit and purpose of the competition (and the reason for offering prizes) is to give our beneficiary organizations the best possible solution *along with working code they can actually use*. In accordance with the competition rules, if we can't get your code working and reproduce your results with a reasonable effort, or if your entry is too disorganized to be practically usable, then your entry may be disqualified!

The overall concept is to **set up this archive as if it were a finished open source project**, with clear instructions, dependencies and requirements identified, and code structured logically with an obvious point of entry. Check out the competition prize winner README template to get started. We also have a data science project template which may be helpful.

At a minimum, your solution must include **an extremely clear README** that details all of the steps necessary to get to your submission from a fresh system with no dependencies (e.g., a brand new Linux, Mac OS X, or Windows installation depending on what environment you choose to develop under) and no other data aside from the raw data you downloaded from us.

This will probably entail the following:

- Necessary tools and requirements (e.g. "You must install Word2Vec 0.1c" or "Install the required Python packages in `requirements.txt`").
    - **All requirements should be clearly documented**, for instance in either a requirements.txt file with versions specified or environment.yml file.
- The series of commands, in order, that would get a reasonably experienced and savvy user from your code to a finished submission.
    - **Ideally, you will have a main point of entry to your code** such as an executable script that runs all steps of the pipeline in a deterministic fashion. A well-constructed Jupyter notebook or R script meets this standard.
    - **The next best thing is a list of specific, manual steps** outlining what to do. For example, "First, open Word2Vec and set these parameters. [...] Take the output file and run the script `src/make_preds.R` with the following parameters [...]". *(The limitations of this approach should be clear to any experienced data scientist!)*
- **Make sure to provide access to all trained model weights necessary to generate predictions from new data samples** without needing to retrain your model from scratch. Note that model weights can be contained in your archive or shared via a cloud storage service. The solution should provide clear instructions to perform inference on a new data point, whether or not it is included in the test set.
- Any other instructions necessary to end up with your winning submission file (or comparable — we understand that certain parts of model fitting are stochastic and won't result in exactly the same parameters every time).

II. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Provide a high level summary of your approach: what did you do and why?
   We train collaboratively a single multi-perceptron neural network regressor over a set of clients, where each client is associated with one of 25 selected airlines. We believe using neural network is a more appropriate choice than boosting trees for federated learning.

2. What ways did you consider adapting your Phase 1 approach to accommodate federated learning? Which did you eventually implement and why?

   We just used almost features adopted in Phase 1 to Phase 2. We dropped one feature regarding the runways since we did not observe it increased the performance, while adding such features can slow down the training procedures. We changed the XGBoost regressor to neural network regressor so that we can adopt the flower federated learning toolbox.

3. Compare the performance (in terms of mean absolute error or other relevant metrics) of your Phase 1 and Phase 2 models.
   We observed a drop in performance of the federated model (MAE ~14 on all labeled data) compared to the centralized model (MAE ~12) in Phase 1 and there are two reasons for this. First, we are using the neural network in Phase 2 which might not be the best for tabular data as boosting trees used in Phase 1. Second, we need to downsample the training data to avoid crashing memory.

4. What experiments did you undertake to optimize performance? Do you believe your final Phase 2 submission achieves the best possible performance of a federated version of your Phase 1 model? If not, what do you think would be needed to achieve the best possible performance? What are the trade-offs involved between performance and privacy for your Phase 2 solution?

   We varied the learning rate, the number of federated rounds and the number of clients selected on each round. The final submission is certainly not the best performance. The reason is because we do not tune these hyper-parameters too much since it requires appropriate privacy preserving tunning otherwise using optimal parameters might leak privacy of each clients' data. I think there is a strong trade-off between privacy and accuracy in our solution. If each client (each airline) is willing to contribute a small validation set and send it to the server, we can improve the performance significantly.

5. What do you see as the most significant remaining work needed to deploy your Phase 2 solution in the real world? For example, coordinating training and/or evaluation, incorporating other private variables, addressing privacy concerns, etc.

   I think deploying some differential privacy mechanisms such as gradient clipping and adding some Gaussian is the most significant future work. Right now, in our current implementation the clients send the naked gradient updates. This certainly poses a privacy risk since it was observed that training data can be reconstructed with high precision given access to the gradient updates.

6. Do you have any useful charts, graphs, or visualizations from the process?

   No.

7. Copy and paste the 3 most impactful parts of code you developed during Phase 2. Explain what each does and how it helped your model.
   The following are 3 most impactful parts of code:

   a. The following line of codes uses the ReLU function to make sure the output of regressor is positive. This is important since we know the value we are trying to predict is always positive. Encoding such prior knowledge slightly improves the overall performance.

```
8.      self.layer_out = nn.ReLU()
```

   b. The following line of codes downsamples the training data to only 2% so that we can train the model on our desktop machine. In practice, if each client's machine is powerful enough it might not be necessary to perform downsampling

```
pd00 = data_dict[airline].sample(frac =SAMPLE_FRAC)
```

   c. The following line of codes use the L1loss (MAE) as the objective function to optimize the network regressor. We observed using L1Loss instead of MSELoss can improve the performance since L1Loss is less sensitive to outlier than MSELoss

```
criterion = nn.L1Loss()
```

9. Please provide the machine specs and time you used to run your model.
   - CPU (model): Apple M1 Pro
   - GPU (model or N/A): N/A
   - Memory (GB): 32 GB
   - OS: Mac OS Ventura 13.4 (22F66)
   - Train duration: 1.5 hour
   - Inference duration: ~30 minutes per whole data of one airport

10. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
    Given our limited computing system, we can train a federated learning model over a very subset of training data. We only select 2% of full data as training data.

11. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

    No

12. How did you evaluate performance of the model other than the provided metric, if at all?

    We did not use any other metric rather than the MAE. The overall performance is a weighted combination of MAE over different airlines where the weights are captured by the training size of each airline.

13. What are some other things you tried that didn't necessarily make it into the final workflow? (quick overview)

    We tried to vary the network architecture such as changing the activation functions or increased the network depth but they did not show significant improvement.

14. Are there any other learnings or conclusions from Phase 2 that are not covered in the previous questions that you would like to include? (optional)