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?

In the phase 2 of the competition, we focused on federating the models built for phase 1. The methodological approach stayed the same – building one CatBoostRegressor per airport predicting the pushback time from 300 synthetically generated features – but now with MFS and Standtime data obfuscated.

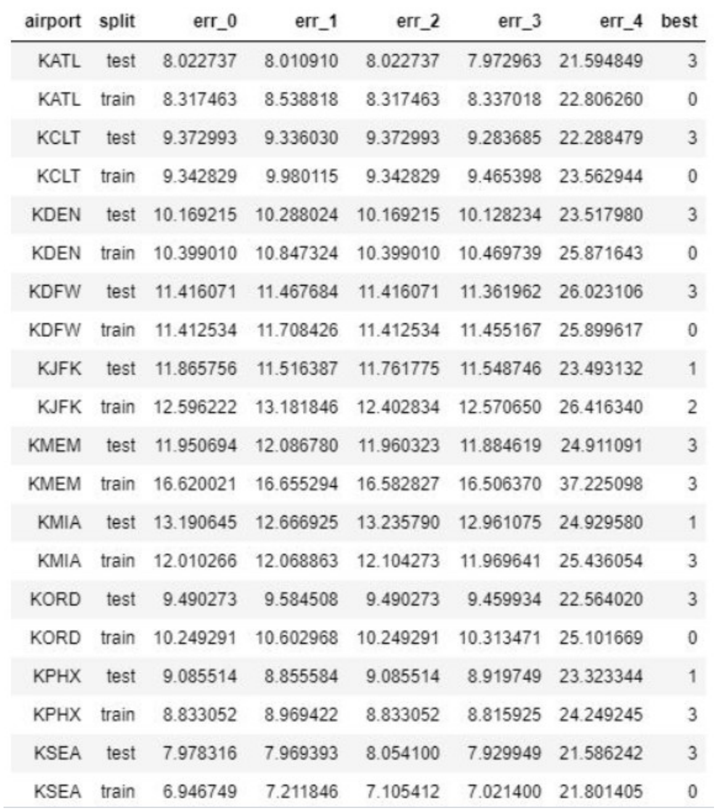
In order to do so, we leveraged the Flower python framework and iterated (defining each airline as client) to update the model weights in a distributed manner.

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

Several approaches were tried like switching to deep learning models or building a model for each combination of airline + airport. But finally we decided to go with one model per airport and one client per airline due to its success in phase 1, and ease to integrate within Flower.

3. Compare the performance (in terms of mean absolute error or other relevant metrics) of your Phase 1 and Phase 2 models.

We can only compare for the train set and our internal validation split's performance since the testing set used in NASA's runtime is private. In those test sets we have observed a decrease in predictive performance from the centralized version of the model to the federated one. For phase 1 we performed a thorough analysis of the errors as shown below with different model approaches showcasing an average MAE across airports in the train set of about 11.5



airport	split	err_0	err_1	err_2	err_3	err_4	best
KATL	test	8.022737	8.010910	8.022737	7.972963	21.594849	3
KATL	train	8.317463	8.538818	8.317463	8.337018	22.806260	0
KCLT	test	9.372993	9.336030	9.372993	9.283685	22.288479	3
KCLT	train	9.342829	9.980115	9.342829	9.465398	23.562944	0
KDEN	test	10.169215	10.288024	10.169215	10.128234	23.517980	3
KDEN	train	10.399010	10.847324	10.399010	10.469739	25.871643	0
KDFW	test	11.416071	11.467684	11.416071	11.361962	26.023106	3
KDFW	train	11.412534	11.708426	11.412534	11.455167	25.899617	0
KJFK	test	11.865756	11.516387	11.761775	11.548746	23.493132	1
KJFK	train	12.596222	13.181846	12.402834	12.570650	26.416340	2
KMEM	test	11.950694	12.086780	11.960323	11.884619	24.911091	3
KMEM	train	16.620021	16.655294	16.582827	16.506370	37.225098	3
KMIA	test	13.190645	12.666925	13.235790	12.961075	24.929580	1
KMIA	train	12.010266	12.068863	12.104273	11.969641	25.436054	3
KORD	test	9.490273	9.584508	9.490273	9.459934	22.564020	3
KORD	train	10.249291	10.602968	10.249291	10.313471	25.101669	0
KPHX	test	9.085514	8.855584	9.085514	8.919749	23.323344	1
KPHX	train	8.833052	8.969422	8.833052	8.815925	24.249245	3
KSEA	test	7.978316	7.969393	8.054100	7.929949	21.586242	3
KSEA	train	6.946749	7.211846	7.105412	7.021400	21.801405	0

In phase 2 we have seen this MAE increase to about 12.3 but with great level of variability from one airport to another and from one run to another.

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?

The experiments performed to optimize performance (both in terms of MAE and runtime of the pipeline), were to change the parameters within the strategy of the Flower framework such as number of batched clients, parallel workers… The final phase 2 solution is worse than the centralized model from phase 1 as reported in question 3, and this could be bridged by building a model with weights being updated with a stratified sample from different airlines at a time (not as in our phase 2 solution where each updates consist of data from only one airline) but with feature obfuscation being done in the central aggregator.

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.

In order to make our phase 2 solution operative in the real world there are several dimensions that would need to be reconsidered and refined accordingly, some of them being:

- ***Value of each feature block**: When building a federated learning model we should first assess the value of each feature block since integrating it into the workflow carries an increased potential problems. In this case we used all the available features to minimize MAE, but this might not be the only metric to optimize for in the real world – such as the tradeoff between MAE and model failures, robustness to feature drift…*
- ***Relevance of the different predictions**: Not all pushback predicitons carry the same business weight (some impact more traffic, have more passangers, carry more risk…) this should somehow be factored into the modelling pipeline to penalize errors asymmetrically for some flights*

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

They have been included in the README markdowns of the repositories shared

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.

3 of the most relevant parts of our code for phase 2 submission are:

- *Feature creation (same as phase 1) (all the code in utilities.py)*
- *Definition of the train client factory needed for federated learning*
- *Main execution of the train pipeline*

Some snippets of the above can be found below:

```
def train_client_factory(airport: str, airline: str):
    """
    Main code for the flower framework to trigger the master creation and
    model training for each client instance - being at the airline level
    """
    try:
        # Load public features for the specified airport and all airlines
        public_features = extract_public_features(airport, train=True)

        # Load private features for the specified airline and airport pair
        private_features = extract_private_features(public_features, airport,
airline)

        # Filter perimeter to stick only to current airline
        perimeter = public_features["perimeter"][public_features["perimeter"]
["gufi"].str[:3] == airline]

        # Combine all features into a master table
        master = build_master(base=perimeter,
                              mfs=private_features["mfs"],
                              config=public_features["config"],
                              etd=public_features["etd"],
                              mom=public_features["mom"],
                              weather=public_features["lamp"],
                              runways=public_features["runways"],
                              standtimes=private_features["standtimes"])

        # Define the train validation split for the current airline slice
        feat_names = [c for c in master.columns if c not in ["timestamp",
"gufi", "minutes_until_pushback"]]
        cat_features = [i for i in range(len(feat_names)) if '_cat_' in
feat_names[i]]
        x_train = master[master["timestamp"] < END_TRAIN][feat_names]
        x_val = master[master["timestamp"] >= END_TRAIN][feat_names]
        y_train = master[master["timestamp"] < END_TRAIN]
["minutes_until_pushback"]
        y_val = master[master["timestamp"] >= END_TRAIN]
["minutes_until_pushback"]

        @dataclass
        class CatBoostRegressionTrainClient(fl.client.NumPyClient):
            """CatBoost regression flower client"""

            airline: str

            def fit(self, parameters=None, config=None):
                model_path = os.path.join(BASE_PATH, "models",
f"{airport}_model")
                previous_model = initialize_model(airport)

                logger.info(f"Fitting airport {airport} with {airline} data")
                logger.info(f"Best iteration of previous model
{previous_model.best_iteration_}")
                logger.info(f"Training with {x_train.shape} and validating on
{x_val.shape}")

                model = CatBoostRegressor(eta=0.01,
                                          depth=7,
                                          rsm=1,
                                          subsample=0.8,
                                          max_leaves=21,
                                          l2_leaf_reg=3,
                                          min_data_in_leaf=50,
                                          n_estimators=5000,
                                          task_type='CPU',
                                          thread_count=-1,
                                          grow_policy='Lossguide',
                                          has_time=True,
                                          random_seed=4,
                                          loss_function='MAE',
                                          boosting_type='Plain',
                                          max_ctr_complexity=12,
                                          bootstrap_type='Bernoulli')

                model.fit(x_train, y_train,
                          eval_set=(x_val, y_val),
                          use_best_model=True,
                          verbose=200,
                          cat_features=cat_features,
                          early_stopping_rounds=20,
                          init_model=previous_model if
os.path.exists(model_path) else None
                         )

                save_weights(model, airport)

                return np.array([[1, 2, 3], [4, 5, 6]], np.int32),
len(y_train), {}

            def evaluate(self, parameters, config):
                logger.info(f"Evaluating client {self.airline}")
                model = initialize_model(airport)
                predictions = model.predict(x_val)
                loss = mean_absolute_error(y_val, predictions)
                logger.info(f"{self.airline} loss = {loss:4.4f}")
                return loss, len(y_val), {}

        return CatBoostRegressionTrainClient(airline)

    except:
        @dataclass
        class DummyTrainClient(fl.client.NumPyClient):
            """Dummy flower client"""

            airline: str

            def fit(self, parameters=None, config=None):

                return np.array([[1, 2, 3], [4, 5, 6]], np.int32), 100, {}

            def evaluate(self, parameters, config):

                return 100, 100, {}

        return DummyTrainClient(airline)
```

```
if __name__ == "__main__":
    """
    Main entrypoint to train and save the model artifact for a given airport
    Example execution: python3.10 train.py "airport_name"
    """

    airport = sys.argv[1]

    client_resources = {"num_cpus": 1}

    strategy = fl.server.strategy.FedAvg(
        fraction_fit=0.1,
        fraction_evaluate=0.1,
        min_fit_clients=len(AIRLINES),
        min_evaluate_clients=len(AIRLINES),
        min_available_clients=len(AIRLINES))
```

```
fl.simulation.start_simulation(
    client_fn=lambda x: train_client_factory(airport, x),
    client_resources=client_resources,
    clients_ids=AIRLINES,
    strategy=strategy,
    config=fl.server.ServerConfig(num_rounds=1),
    ray_init_args={"include_dashboard": False,
                   "num_cpus": 1
                  },
)
```

8. Please provide the machine specs and time you used to run your model.
   - CPU (model): Intel Core i7 CPU @1.9GHz
   - GPU (model or N/A): N/A
   - Memory (GB): 32Gb of local RAM memory
   - OS: Windows 10
   - Train duration: Average of 7h per airport with some variance
   - Inference duration: Same as phase 1, 7h for the 1800 timestamps in the held-out test set if run on the NASA runtime since only model weights change and most of the time comes from the feature engineering part

9. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

Following the README file should suffice to fully reproduce the training and inference pipeline

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

No, the final submission is self-contained

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

MAE of the entire data sample was used as a goodness of fit indicator for pushback time estimation, and MAE was also analyzed at a breakdown level for each airport and month to gain better insight into where the model was performing poorly

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

As mentioned in 2, several approaches were tried among them switching to deep learning (ease with Flower, but difficulty with categorical features, need for normalization…) – And also a model *for each pair of airport and airline was tried but due to small data for some instances and lack of generalizability it was dropped in favor of the submitted solution.*

13. 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)

Yes, thanks to federating our solution from phase 1 we have gained a deeper insight into what it takes and can be done when using Flower. Some insights being:

- *Federated learning is needed for some contexts where data privacy is a high priority*
- *Some very relevant features that might help decrease MAE are private to each airline*
- *In order to federate our models we need to iterate different model versions (in the case of our approach weights) across clients to not leak private information from one airline to another*